A NOVEL LEARNING METHOD BY STRUCTURAL REDUCTION OF DAGS FOR ON-LINE OCR APPLICATIONS

I-Jong Lin and S.Y. Kung

Dept. Of Electrical Engineering, Princeton University
Princeton, NJ

ABSTRACT

This paper introduces a learning algorithm for a neural structure, Directed Acyclic Graphs (DAGs) that is structurally based, i.e. reduction and manipulation of internal structure are directly linked to learning. This paper extends the concepts in [1] for template matching to a neural structure with capabilities for generalization. DAG-Learning is derived from concepts in Finite State Transducers, Hidden Markov Models, and Dynamic Time Warping to form an algorithmic framework within which many adaptive signal techniques such as Vector Quantization, K-Means, Approximation Networks, etc., may be extended to temporal recognition. The paper provides a concept of path-based learning to allow comparison among Hidden Markov Models (HMMs), Finite State Transducers (FSTs) and DAG-Learning. The paper also outlines the DAG-Learning process and provides results from the DAG-Learning algorithm over test set of isolated cursive handwriting characters.

1. INTRODUCTION

The workhorse for temporal recognition problems has been the Hidden Markov Model. In cursive handwriting and in speech recognition, HMMs are a popular and high quality solution [2]. However, from a perspective of structural integration, the current HMM Learning theory accepts only single-path structures (vs. multi-path graphs) and is restricted by its inability to expand or reduce its own structure.

Finite State Transducers (FSTs) [3] have these structural capabilities, manipulating complex multi-path input and optimally reducing structure through a state minimization algorithm. In recognition problems, FSTs are primarily used in grammar modeling, word-lattices and other high-level constructs. However, the FSTs are less applicable to general signal processing because the current domain of FSTs is restricted to the discrete, finite-element languages.

DAG-Learning has much in common with both HMMs and FSTs. While HMM learning is a parametric optimization technique where structure is optimized implicitly, DAG-Learning optimizes structure and locally optimizes parameters once structure is set. This change allows DAG-Learning to be compatible with multi-path segmentation schemes [1].

	Input to Model	Solution	Drawbacks
HMM	Sequences (single path input)	Locally optimal general solution	Single Path Restrictions, Fixed Size
FST	Multi-path input, discrete finite-element language	Optimal Solution	Explicit Quantization
DAGs	Multi-path finite-length input (acyclic)	Heuristic acyclic solution	Time/Size Inefficiency

Table 1: Comparisons between HMM, FST and DAG-Learning

Given a partial order, DAG-Learning synchronizes data from multiple sources and different segmentations and subsequently compresses the synchronized segments. DAG-Learning extends the concepts of FSTs to a broader concept of similarity over real value signals (see Table 1).

1.1. Path-Based Learning

The common link among the three models of HMMs, FSTs and DAGs is that the path [1] can be considered as the basic unit of learning. In each case, the process of learning creates a graph whose paths describe a set of inputs graphs and in which criteria of accuracy, retrieval time and storage requirements are optimized. The three learning algorithms can be stated optimizations of a path-based problem as follows (see Table 2):

Let there be function $\mathcal{S}$ that measure similarity between paths (denoted as $\overset{\rightsquigarrow}{p}$). The similarity function should be reflexive and symmetric. An important metric is how two

[1] Given a graph with two distinguished nodes, a source and a sink, a path in this paper corresponds to the subgraph of nodes and edges that lead from source to sink.

paths compares with $\mathcal{S}$ to an unknown input. Under the conditions that similarity function is well-behaved, it is assumed that, for a given error value ϵ,

$$\forall\ \vec{\tilde{x}}, |\mathcal{S}(\vec{\tilde{x}}, \vec{\tilde{p_1}}) - \mathcal{S}(\vec{\tilde{x}}, \vec{\tilde{p_2}})| \leq \epsilon \Rightarrow \exists \lambda, \mathcal{S}(\vec{\tilde{p_1}}, \vec{\tilde{p_2}}) \geq \lambda \quad (1)$$

Definition 1 (Path-containment) *Let there be graphs, G_1 and G_2. G_1 is said to path-contain G_2 ($G_1 \vec{\tilde{\subset}} G_2$) if, for a given λ,*

$$\forall\ \vec{\tilde{p_1}} \in G_1, \exists\ \vec{\tilde{p_2}} \in G_2,\ \textit{s.t.}\ \mathcal{S}(\vec{\tilde{p_1}}, \vec{\tilde{p_2}}) \geq \lambda \quad (2)$$

Definition 2 (Path-equivalence) *Let there be graphs, G_1 and G_2. G_1 is said to be path-equivalent G_2 ($G_1 \vec{\tilde{=}} G_2$) if, for a given λ,*

$$(G_1 \vec{\tilde{\subset}} G_2) \wedge (G_2 \vec{\tilde{\subset}} G_1) \quad (3)$$

Note that path-equivalence is not graph isomorphism.

Definition 3 (Path-Learning Problem) *For a given similarity function and a parallel network of input graphs, the path learning problem is to find a graph G with least number of nodes and/or edges that is path-equivalent to G.*

Testing for Path-Containment or Path-Equivalence and the Path-Learning problem itself are all NP-hard (simple reductions to 3SAT).

HMM and the other derived HMMs (such as autoregressive HMMs) solve this problem by starting with a fixed structure and modifying its parameters to maximize the expected log probability of state sequences to match the sequences seen in the test set.

Finite State Transducers minimize their structure by exponentially expanding the graph, but optimally minimizing the expanded graph in linear time. Since elements of the input language are finite and discrete, their similarity function is an equivalence relation.

DAG-Learning borrows many concepts from the FSTs, but places the learning algorithms over a different domain, sequences and/or acyclic multi-path graphs of real values. On a real-valued domain, it is unlikely that $\mathcal{S}$ is transitive,

$$(\mathcal{S}(\vec{\tilde{p_1}}, \vec{\tilde{p_2}}) \geq \lambda) \wedge (\mathcal{S}(\vec{\tilde{p_2}}, \vec{\tilde{p_3}}) \geq \lambda) \not\Rightarrow \mathcal{S}(\vec{\tilde{p_1}}, \vec{\tilde{p_3}}) \geq \lambda \quad (4)$$

and therefore $\mathcal{S}$ is not an equivalence relation. Without this vital characteristic of transitivity and an infinite element language, the current FST minimization is not applicable.

2. DAG-LEARNING

With DAG-Compare as its retrieval function, DAG-Learning is a subset of graph optimization techniques for path-learning problem that attempt to maintain path-equivalence. Although graph optimization can accept any graph-to-graph related reduction transforms [4], DAG-Learning divides its graph transformations into separate node and edge reductions. DAG-Learning can be considered a divide-and-conquer technique that first synchronizes the segments of different examples (node reduction) and then uses standard data compression techniques on sets of synchronized edges to reduce their number (edge reduction).

Learning Algorithm	Optimization Technique for the Path-Learning Problem
Hidden Markov Model	Maximization of λ through EM maximization of expected log probability over the set of sequences
Finite State Transducer	Optimal reduction (exponential in time and space) under language restriction
DAG-Learning	Optimization of structure through graph transformations that maintain path-equivalence

Table 2: Learning algorithms framed within the path-learning problem

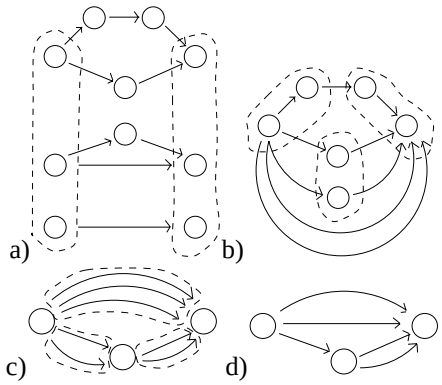

Figure 1: Steps in DAG-Learning with the dashed lines representing reducible sets a) Set of exemplars with initial structure to be formed b) initial structure with reducible nodes c) reduced node structure with reducible edges d) final DAG-Learned structure

2.1. The Initial Structure

As shown in figure 2 a, the initial structure for DAG-Learning is a parallel network composed of exemplars. A classifier that uses this initial structure implements Nearest Neighbor classifier with the DAG-Compare as distance criterion. As with the nearest neighbor recognizer, the space requirements limit its practicality and motivates reduction through DAG-Learning.

2.2. Node Reductions

As shown in figure 2 b, DAG-Learning next identifies reducible nodes, i.e. nodes whose merger do not violate the path-equivalence between the original graph and its reduced

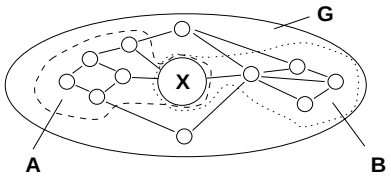

Figure 2: Induced graphs of graph G w.r.t. A is the upstream graph; B is the downstream graph.

version. In physical terms, node reducibility corresponds where physical breaks in the data stream can be synchronized w.r.t. the partial ordering imposed by the graph. This ordering is determined by their relative position of the segments which come before and after the break in partial order. In the acyclic graph, this ordering induces two subgraphs relative to the node that corresponds to the break, the upstream and downstream graph: one from all the paths from source to the node and the other from all paths from the node to the sink, respectively (see Figure 2). A family of node reductions is based upon path-containment of these subgraphs.

For complete graphs, G_1 and G_2, path containment is calculated from Eq. 2 and the definition of the DAG-Compare operation:

$$\min_{\vec{p_1} \in G_1} \left(\max_{\vec{p_2} \in G_2} \mathcal{S}(\vec{p_1}, \vec{p_2}) \right) \geq \lambda \tag{5}$$

$$\min_{\vec{p_1} \in G_1} (DAG\_Compare(\vec{p_1}, G_2)) \geq \lambda \tag{6}$$

To calculate these path containment over induced subgraphs, a similarity score over partial paths must be defined. Using the recurrence relation form for the similarity score as dictated by [1] and a function $\phi(X_1, X_2)$ that estimates the partial score between all paths that end at node X_1 and all paths that end X_2, we can now define a partial similarity score $\mathcal{S}'$, for a given pair of node X_1, X_2 as follows. Let $\vec{x_1}$ be the path along $\vec{p_1}$ from node X to the sink, similarly for $\vec{x_2}$.

$$\mathcal{S}'(\phi(X_1, X_2), \vec{x_1}, \vec{x_2}) = \mathcal{S}(\vec{p_1}, \vec{p_2}) \tag{7}$$

With this definition and by modifying the basis of the DAG-Compare operation to accept any starting value, upstream path-containment is defined as:

$$\min_{\vec{a_1} \in A} DAG\_Compare(\vec{a_1}, A_2, \phi(src., src.)) \geq \phi(X_1, X_2) \tag{8}$$

Downstream path-containment is defined as:

$$\min_{\vec{a_1} \in A} DAG\_Compare(\vec{a_1}, A_2, \phi(X_1, X_2)) \geq \lambda \tag{9}$$

The function ϕ is not known, but in practice can be approximated well.

With these definitions, three node reductions are listed below. Two nodes in a graph, X_1 and X_2 whose upstream and downstream graphs are $A_1(A_2)$ and $B_1(B_2)$, respectively, can be merged together if they satisfy any of these three path-containment conditions, using Eqs.8 and 9:

$$\textit{Upstream Equiv.:} \quad A_1 \overset{\rightsquigarrow}{=} A_2 \tag{10}$$

$$\textit{Downstream Equiv.:} \quad B_1 \overset{\rightsquigarrow}{=} B_2 \tag{11}$$

$$\textit{Subsumption:} \quad (A_1 \overset{\rightsquigarrow}{\subset} A_2) \wedge (B_1 \overset{\rightsquigarrow}{\subset} B_2) \tag{12}$$

$$\textbf{or} \quad (A_2 \overset{\rightsquigarrow}{\subset} A_1) \wedge (B_2 \overset{\rightsquigarrow}{\subset} B_1) \tag{13}$$

When up/downstream path-containment and possible node reductions have been calculated, the optimal reducible sets of nodes must be found. This problem reduces to the NP-complete problem of graph coloring. For our application, we use a simple greedy algorithm for our application. Subsumption merging reduces to the NP-complete problem of node covering. Once again, for our application, we use a greedy algorithm.

Over the last four paragraphs, we have identified three separate NP-complete problems for DAG-Learning. In practice, heuristic solution can be used to arrive a non-optimal, good quality solution (see Section 3).

2.3. Edge Reduction

As shown in figure 2 c, after node reduction, a set of edges may share the same starting and ending nodes. These edges are synchronized w.r.t. the partial ordering of the segmentation. Edge reduction compresses the synchronized data by representing multiple vectors with a fewer vectors. Using the inverse of the similarity function as distance, edges can be reduced or modeled through a number of data compression techniques such as Vector Quantization, K-means, approximation networks, Gaussian distribution, etc.

3. RESULTS

A simple DAG-learning algorithm which uses greedy algorithms for coloring and VQ for edge reduction was tested on a set of DAG-coded isolated cursive lowercase letter characters from ten different writers. The processing of data is described in [1]. To ensure that DAG-Learning has basic characteristics of a learning algorithm, the relationship between the size of training set and their resulting DAG-Learned network must be established.

The first characteristic of a learning algorithm is that the the network converges if given an infinite amount of information about a process. HMMs reach a local optimal through the Baum-Welch learning rules and FSTs reach a global optimal size through learning. Currently, we cannot prove whether the size of learned network will converge. Results in figure 3 show empirically that DAG-Learning asymptotically reaches a fixed size (dependent on λ) as the number of examples increase.

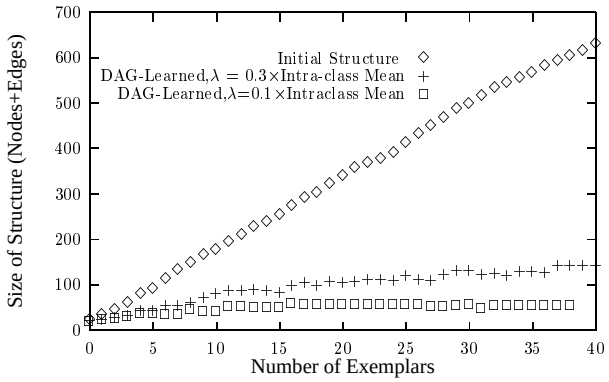

Figure 3: Structural Convergence in DAG-Learning: Note that, on a increasing set of lowercase a's from single writer, the untrained structure increases linearly while the DAG-Learned structure reaches an asymptote dependent on λ.

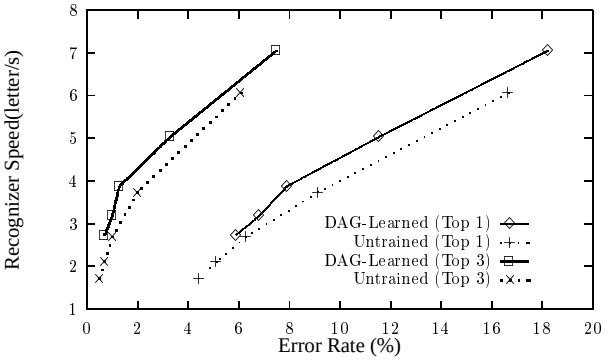

Figure 4: Speed vs. Accuracy Trade-off: Results are writer-dependent recognizer. The points on the line count off the number of exemplars per class used, starting from right to left, going from 1 to 5.

The second test is to show that learning improves the accuracy vs. speed trade-off over a Nearest Neighbor type rule. Results in figure 4 is from a writer-dependent isolated cursive letter recognizer over ten different writers, using 1 to 5 exemplars per writer. Results show that the speed vs. accuracy curve always favors DAG-Learned recognizer over the untrained recognizer, even with the small number of exemplars.

The final test is to show that the network generalizes data such that comparison of different types of examples is invariant to writer style, noise and other distortions. The writer-independent isolated cursive letter recognizer takes its training set from six writers with 65 exemplars per letter. The test set is from four unseen writers with 60 of examples per letter. Results in table 3 show a 4.0% loss in top 1 accuracy, 0.1% loss in top 3 accuracy, but the speed of the DAG-Learned recognizer is 2.76 of the untrained.

	Accuracy (Top 1/Top 3)	Speed (letter/s)	Size $\|\|V\|\| + \|\|E\|\|$
Initial Structure	85.1% / 96.4%	0.086	29894
After DAG-Learning	81.1% / 96.3%	0.238	11642

Table 3: Results from a Writer-Independent Isolated Cursive Letter Recognizer, using 6 different writers with 65 exemplars per class. Test set was 4 unseen writers, with 40 exemplars per class. Note DAG-Learned recognizer is 176% faster than the untrained one.

4. CONCLUSION

DAG-Learning is a novel learning algorithm that blends signal processing techniques with the FSM minimization.

DAG-Learning and HMM training are also both complementary and orthogonal optimization techniques, given a few minor modifications to each side. Once the DAG-structure has been fixed, E-M learning can be applied to maximize the edge parameters. Considering only left-right (Bikel) Markov models, DAG-Learning can initialize structure or form an inter-class training by re-integrating failed sequences for retraining.

This paper completes the basic algorithmic kernel for DAG computation and gives adaptability to the framework of the DAG recursive structure [5]. Our future research will explore the representation and comparison of multidimensional input (text, images, video, etc.), using multi-axial segmentation DAGs.

5. REFERENCES

[1] I-Jong Lin and S.Y Kung. Coding and comparison of dags as a novel neural structure with applications to on-line handwriting recognition. *To appear in IEEE Transactions in Signal Processing Special Issue Neural Networks*, 1996.

[2] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–86, February 1989.

[3] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23, 1997.

[4] Y. Le Cunn, L. Bottou, and Y. Bengio. Reading checks with multilayer graph transformer networks. In *Proceedings of ICASSP 1997*, volume 1, page 151, 1997.

[5] I-Jong Lin and S.Y Kung. A recursively structured solution for handwriting and speech recogntion. In *Proceedings of 1997 IEEE Workshop on Multimedia Signal Processing*, pages 587–592, 1997.