Modeling Online Sketching as a Dynamic Process

Tevfik Metin Sezgin MTSEZGIN@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory, 32-235 Vassar st., Cambridge MA, 02139 USA

Abstract

Online sketching is an incremental and dynamic process; sketches are drawn over time, one stroke at a time, and can be captured with devices such as Tablet PCs and pen based PDAs. We have shown that the dynamic properties of the sketching process contain valuable information that can aid recognition. We describe a framework that can handle complex user input. Specifically, we show how we can take advantage of the regularities in sketching even when users draw objects in an interspersed fashion.

1. Introduction

Online sketching is an incremental and dynamic process: sketches are drawn one stroke at a time and be captured in devices such as Tablet PCs and pen based PDAs. This is unlike scanned documents or pictures which only capture the finished product. The dynamic properties of the sketching process contain valuable information that can aid recognition (Sezgin & Davis, 2005). In particular, in a number of domains the order in which users lay out strokes during sketching contains patterns and is predictable. We have presented ways of taking advantage of these regularities to formulate sketch recognition strategies (Sezgin & Davis, 2005). Here, we describe a framework that can handle more complex user input. Specifically, we show how we can take advantage of the regularities in sketching even when users draw objects in an interspersed fashion (e.g., start drawing object A, draw B before fully completing A, come back and complete drawing A).

2. Sketching as a stochastic process

Previous work has shown that in certain domains, stroke ordering follows predictable patterns and can be modeled as a Markovian stochastic process. Work in (Sezgin & Davis, 2005) shows how sketches of mechanical engineering drawings, course of action diagrams, emoticons and scenes with stick-figure can be modelled and recognized using Hidden Markov Models. In these domains, HMM-based modeling and recognition is possible because objects are usually drawn one at a time using consistent drawing orders. The HMM-based approach exploits these regularities to perform very efficient segmentation and recognition.

The HMM-based recognition algorithm scales linearly with the scene size, but requires each object to be completed before the next one is drawn. In certain domains, although there is a preferred stroke ordering, objects can be drawn in an interspersed fashion. For example, in the domain of circuit diagrams, people occasionally stop to draw wires connected to the pins of a transistor before they complete the transistor. One way of thinking about such a drawing scenario is that, instead of a single Markov process, we have multiple processes that generate observations, and the task is to separate observations from these processes. We model such drawing behavior as a multimodal stochastic process that can switch between different individual Markov processes, each of which captures drawing orders for individual objects. Although the new approach can also be described as a HMM, it is more easily described and understood using its dual representation as a dynamic Bayesian net (DBN).

Our approach to modeling interspersed drawing behavior is general enough to allow an arbitrary number of objects in a domain to be drawn in an interspersed fashion, but in practice people usually intersperse at most two objects. For example, in the circuit diagrams, unlike other circuit components, transistors have three connection points (emmiter, collector, base), and sometimes people draw the wires connecting to these points when the transistor is only partially drawn, causing interspersing of transistor and wire strokes. We have created a model specialized to handle interspersing of wires with other components in circuit diagram sketches. [1]

3. The network structure

Next we introduce our DBN model for circuit diagrams which handles interspersed drawing orders while still allowing polynomial time inference in the number of strokes.

We model the circuit diagram sketching process using a

[1] Although it is also possible to have a model general enough to allow interspersing between any two objects, we use this specialized model due to the nature of interspersing in our domain.

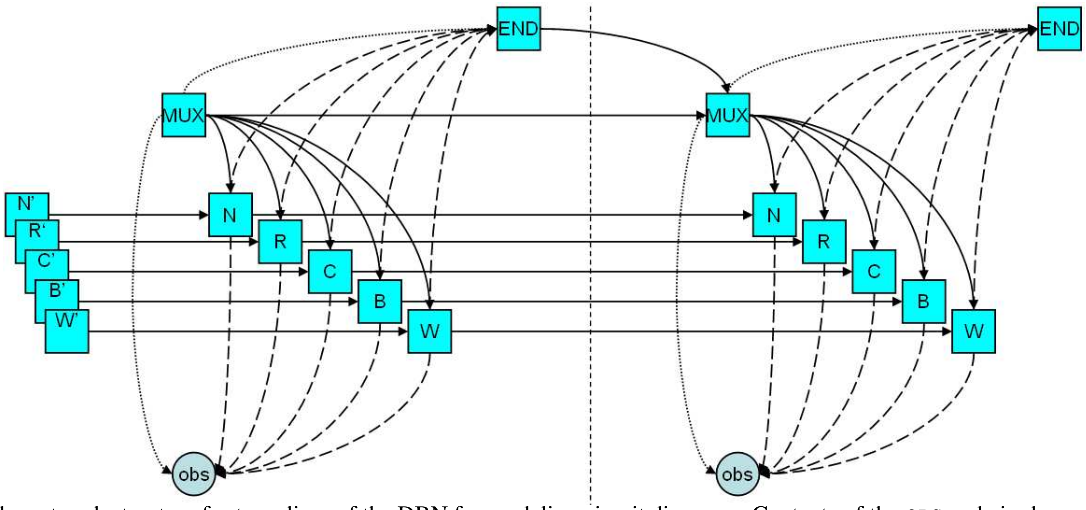

Figure 1. The network structure for two slices of the DBN for modeling circuit diagrams. Contents of the OBS node is shown in Fig. 2.

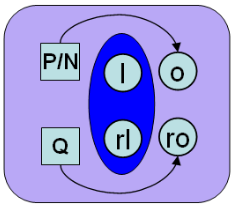

Figure 2. Details of the observation node. L and RL are Gaussian nodes that capture length and relative length. O and RO capture orientation and relative orientation. P/N and Q are mixture parameters for the relative features.

DBN (Fig. 1). The square nodes are discrete and the circular nodes are continuous. All nodes except the OBS node are hidden. The observation node OBS captures a number of features computed using the properties of primitives derived from strokes and the details of this node is shown in Fig. 2.

The hidden nodes in Fig. 1 and their connections specify the generative process that models the way in which objects in the domain are drawn. In our domain, we have five objects: NPN transistors, resistors, capacitors, batteries, and wires. Nodes N, R, C, B and W model the way in which these objects are drawn. Based on the value of the MUX node, only one of these processes is activated. The END node is simply a binary variable that species whether the latest observation completes drawing of the currently active object.

4. Node descriptions

We now describe each node in detail. We will adopt the generative process view of the model and describe the dynamics of the model from that perspective, but the reader should keep in mind that the model is used for assigning probabilities to series of observations obtained by encoding sketches (inference) and not for generating observation sequences.

4.1 The MUX variable

MUX keeps track of the main object the user is drawing (which can be interspersed with wires if it is a transistor). The actual observables are generated based on the value of this node and the individual object process nodes (i.e., N, R etc.). As a result, if there are N different objects that the user can draw, then the MUX node has N states. In addition, this node enters a special state when a pair of objects are being interspersed. There is a unique state for each pair of objects that can be interspersed. In our case, because only wires can be interspersed with transistors, there is only one such state. This is the state that we enter when the user starts drawing wires in the middle of a transistor and entering this state serves as a reminder that after the wires are drawn, we should complete that transistor that we initially started. So the MUX state has $N+1$ states (N for individual objects and one special state for interspersing wires with transistors).

MUX_{t+1} is conditioned on MUX_t and END_t. The reasoning behind this conditioning is twofold: if there is no wire/transistor interspersing, the user may start drawing a new object only if the previous object is completed, and the probability of drawing a particular class of object may depend on the type of the last object.

4.2 The object variables (N, R, C, B and W)

These variables capture how individual objects are generated. In isolation, each node captures the state transition dynamics for an object, and when paired with the observation node, each node can be seen as an HMM that can generate features for that object. These nodes can change

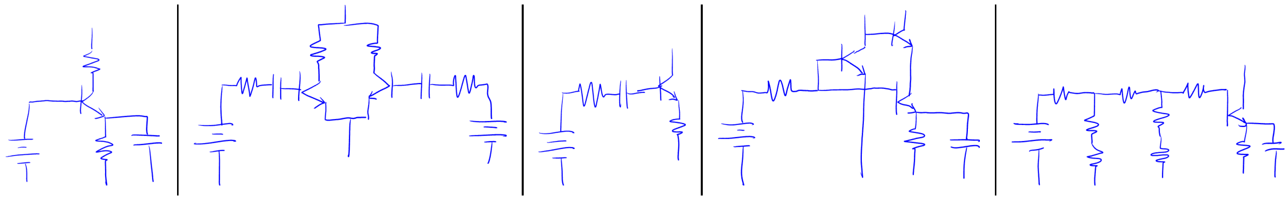

Figure 3. Examples of sketches

state only if they are active (as indicated by the MUX node). For example, the R node can change state only if the MUX node is in its *resistor* state; and in order to handle interspersing the W node can change state only if the MUX node is in its *wire* or *the user is interspersing wires with transistors* states. State transitions are Markovian, with each node conditioned on its value from the previous time slice. Finally the initial frame contains uniform object priors for the object nodes (N', R', C', B' and W')

4.3 The END variable

This discrete binary node is conditioned on one of the object variables, (i.e., at any time, only one of the arcs coming from the object variables is active). The choice of which parent is active is governed by the value of MUX. This is an example of the *switching parent* mechanism of DBNs that we use. The dotted arrow from MUX to END indicates that MUX is the parent that controls the switching behavior for parents of END, and only one of the the dashed arrows from the object variables to END is active based on the value of MUX.

4.4 The OBS variable

This is another example where we have switching parent behavior in our model. As in the END variable, the OBS variable is conditioned on only one of the object variables as determined by the value of the MUX variable. The details of the OBS node is shown in Fig. 2 (P/N, Q, L, RL, O and RO). L and RL are Gaussian nodes that capture length and relative length. O and RO capture orientation of the current primitive and the relative orientation with respect to the previous primitive. O and RO are continuous variables modelled as mixtures of Gaussians, while P/N and Q are the mixture variables that respectively model positive/negative slope for O and the quadrant of the current primitive with respect to the previous primitive for RO.

5. Implementation and results

In order to test our model, we collected circuit diagrams from electrical engineers. We asked users to draw multiple instances of circuits shown in Fig. 3. We collected ten examples of each circuit for a total of fifty circuits. Our current results are limited to data from a single user, but we believe they serve as a proof of concept.

5.1 Training

In order to train and test our model, we labeled the data by assigning object labels to groups of strokes. During training, in addition to the original observable node OBS, the values of MUX and END nodes were supplied, thus the only hidden nodes were the individual object nodes.

5.2 Classification

Once the model is trained, classification is performed by computing the most likely assignment to the MUX variable in each frame for the observation sequence derived from a given sketch.

5.3 Recognition performance

We used instances of the first four circuits in Fig. 3 as training examples and tested our system on all instances of the last circuit in Fig. 3. With only one example of circuits 1-4, we obtained a 73% correct classification rate. Using two and three examples of circuits 1-4 resulted in 89% and 93% correct classification rate consecutively. In cases where there were interspersing, we were able to detect interspersing 67% of the cases.

These results suggest that even as few as four examples can yield good recognition rates, and increasing the number of training examples results in better recognition rates even if the examples come from the same circuit. In addition the results suggest that the conceptual mechanism required to detect interspersed drawing works. We expect the recognition performance to get better with more training data and more data collection is currently underway.

Research Support

This research is funded by the MIT iCampus project and Project Oxygen.

References

Sezgin, T. M., & Davis, R. (2005). HMM-based efficient sketch recognition. *International Conference on Intelligent User Interfaces, San Diego CA January 2005.*