Transfer in Learning by Doing

Bill Krueger, Tim Oates, Tom Armstrong
University of Maryland Baltimore County
CSEE, 1000 Hilltop Circle
Baltimore, MD 21250
{wkrueg1,oates,arm1}@cs.umbc.edu

Paul Cohen, Carole Beal
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{cohen,cbeal}@isi.edu

Abstract

We develop two related themes, *learning procedures* and *knowledge transfer*. This paper introduces two methods for learning procedures and one for transferring previously-learned knowledge to a slightly different task. We demonstrate by experiment that transfer accelerates learning.

1 Introduction

Procedures are interesting for many reasons: Infants exercise procedures (called "circular reactions" by Piaget) almost from birth. These gradually develop in complexity, adding new elements, and eventually incorporate objects in the infant's environment. Piaget believed infants learn much about the world by executing procedures. Older students do, as well; indeed, the literature on expert/novice differences can be summarized in a single phrase: Novices know "how," experts also know "why." Novices can run statistical tests but not understand why they are appropriate, they can follow recipes but not understand the underlying chemistry and gastronomy.

There are advantages to learning fact-like knowledge alongside procedures. One can do useful work with procedures even if one doesn't completely understand them. This means learning is grounded in activity and is gradual over the life of the agent. By exercising procedures the learner produces occasional failures as well as the context one needs to gradually learn both new procedures and non-procedural knowledge. The latter includes the facts and reasons we call "understanding," or, less colloquially, the conditioning variables that affect the probabilities that procedures will succeed. In social environments, there often is a human to help the learner correct missteps before a procedure goes completely wrong. This minimizes the credit-assignment problem.

Our intention is to have a machine learn a sequence of increasingly-difficult games, starting with extremely easy ones. All the games are two-person card games. One player makes mistakes, the other is a competent player who offers corrections as necessary. For convenience we call these players the learner or child (C) and the adult (A). Correct play for the games can be modeled by finite state machines, so the learner learns finite state machines and uses its current machine to generate its next move.

The experiments in this paper involve a sequence of three games. In the first game, the child and the adult each have two stacks of cards, one in which all of the cards are face-up and another in which all of the cards are face-down. Let's denote these stacks by the owner (`A` for adult or `C` for child) and whether they are face-up (`U`) or face-down (`D`). Therefore, the four stacks are `AU`, `AD`, `CU`, `CD`. Each player must flip over the card on top of their face-down stack and move it to the top of their face-up stack. Each player does this as fast as they can, and can ignore the other player's actions for the purposes of this game.

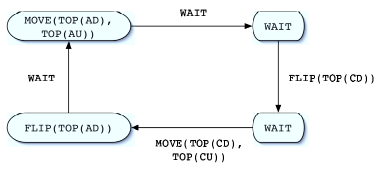

Figure 1: A state machine for Game 2 that yields perfect play.

In the second game, the players take turns flipping and moving cards, as shown in Figure 1. The two nodes on the left specify constraints on what the child observes, namely, the action that is currently being taken by the adult. To be in the lower-left state in the machine, the adult must currently be flipping over the top card on her face-down stack (`FLIP(TOP(AD))`). When this happens, the action executed by the child is `WAIT`. That is, when the adult does `FLIP(TOP(AD))`, the child does `WAIT` *at the same time*. This leads to a state in which the adult does `MOVE(TOP(AD), TOP(AU))`, which also results in the child waiting. Then, when the adult executes a `WAIT` action (the constraint in the upper-right state), the child flips the card on top of her face-down stack, and the adult continues to wait (the lower-right state) as the child does `MOVE(TOP(CD), TOP(CU))`. Some time later, the adult flips the card on top of her face-down stack, and the cycle repeats.

The third game is just like the second game, except when either player turns over a card and the color of that card matches that of the card atop the other player's face-up stack, both players say "Squawk".

2 Learning the State Machines

Our goal is to learn procedures, and, specifically, perfect-play state machines such as those we described in the previous section. A useful distinction can be drawn between learning a state machine and learning a state machine given a related state machine. We are most interested in the second case, which we present as a kind of *transfer learning*. Even so, we begin our discussion of learning with two methods for learning state machines *de novo*. These are Bayesian Model Merging (BMM) [Stolcke, 1994], a well-known method for learning HMMs; and State Splitting (SS), which also learns HMMs, though by splitting rather than merging states.

The training data in both approaches was a sequence of observations such as `MOVE(TOP(CD), TOP(CU))`, the symbol `SQUAWK`, and the special symbol `NO`. The BMM and SS algorithms rely on `NO` to indicate that the child has done something that is not allowed in perfect play. Incorrect play is generated by the learner's incorrect machine in the active learning regime. Passive learning requires training data that includes mistakes and adult admonitions (i.e., `NO`) and corrections. For this purpose we build simulators of child and adult play.

BMM is a top-down approach in which an initial HMM is constructed with one state for every observation, and observations are greedily merged to maximize the posterior likelihood of the data using a description length prior on HMMs. It is incremental in the sense that new observations can be added during the merging process. SS is a bottom-up approach that starts with one state that matches all observations, and repeatedly splits states on observable features of the world in an effort to more accurately predict negative feedback from the teacher. Features are chosen that minimize entropy in the distribution of feedback (positive or negative) over all states.

Regardless of whether the HMM was learned via BMM or SS, it can be used for action selection (game play) and to parse new observations. We are interested in whether learning a series of related, yet increasingly difficult, games requires less effort in total than learning the most difficult game *de novo*. For example, is it easier to learn our three games in sequence, using the machine learned for game n to bias the learning of game $n + 1$, than to learn to play Squawk with no prior machine to serve as bias?

When using BMM to learn a new game biased by an existing HMM, new states and transitions are added for the new observations, but old and obsolete state transitions are not removed. Instead, as more and more data are collected for the new game, the probabilities of obsolete state transitions that are no longer traversed will tend to zero. This is not acceptable, as we want to quickly refine the model to work with the new game while harnessing as much information as possible from previous experience. To achieve this we preserve model structure but eliminate the bias of old probability parameters when moving from one game to the next. In BMM terminology, this means that we reset Viterbi counts to be uniform and small when starting to learn a new game.

3 Experiments

Figure 2 shows two machines that were intermediate steps on the path to learning the turn taking game. In the top machine, there is a single state from which the child can `WAIT`, `MOVE(CD CU)`, or `FLIP(CD)`. Edges are labeled with two counts - the number of times the transition was taken and the number of times that transition resulted in a `NO`. This machine does not yield perfect play, so the state will be split. The observable feature that most reduces entropy in the distribution of negative feedback is the feedback received for the move just made.

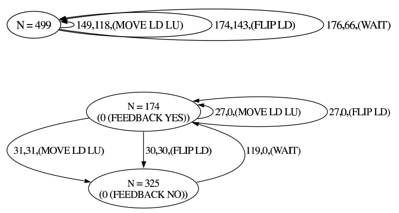

Figure 2: Machines produced by SS when learning the turn taking game.

The bottom machine in Figure 2 shows the result of splitting on this feature and pruning away any transitions that always cause a `NO`. This machine correctly captures the fact that if the adult corrects the child, the only legal action is to `WAIT` while the adult repairs the incorrect action, which will result in a `YES`. Residual feedback non-determinism exists in the top state due to errors in action ordering (e.g., flipping twice in a row). Therefore, that state will be split next.

In a second set of experiments, the child chose actions randomly from a set of known actions until a model was developed (using BMM) that could guide action selection. Initially, therefore, the child would frequently choose an incorrect action. Learning early on with no model is tedious and time-consuming as it requires heavy exploration. The second game, the simple turn-taking game, was still easy to learn. Only 10 simulation time steps were required to learn the optimal model for that game when starting from scratch. Without using this model to bias the model learned for the third game, 53 time steps were required to achieve the optimal model for the more complex game. However, when the game 2 model was used to bias the learning of game 3, only an additional 17 time steps were needed to learn game 3, making a total of 27 time steps. That is, game 3 took in total half as long to learn when game 2 was learned as an intermediate step in the learning process.

Acknowledgments

This work was supported by DARPA (contract number: 53-4540-0016; project name: Learning by Doing).

References

[Stolcke, 1994] Andreas Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California, Berkeley, 1994.