Exploring Supervised Many Layered Learning as a Precursor to Transfer Learning

By

Jeffrey James Juozapaitis

A Thesis Submitted to The Honors College

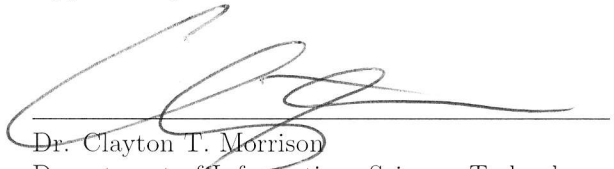
In Partial Fulfillment of the Bachelors degree With Honors in

Computer Science

THE UNIVERSITY OF ARIZONA

DEC 2012

Approved by:



Dr. Clayton T. Morrison
Department of ~~Information~~: Science, Technology, and Arts
Department of Computer Science

Abstract

In this paper, we learn a simple conceptual card game as learned by David Stracuzzi's Cumulus algorithm. We then posit a (sadly unimplemented) scheme to transfer the neural net created by it to a similar game with small modifications, hopefully cutting down the learning time. We then analyze the flaws with the transfer scheme and posit other schemes that may produce better results.

1 Statement of Purpose

The purpose of this project was to add to the field of inductive transfer, or transfer learning. The idea was to learn a simple game, and then find a way to use this old knowledge to improve the learning rate and quality of a more complex game.

2 Statement of Relevance

My field is Computer Science, and Artificial Intelligence (and thus, by proxy, Machine Learning) is a subfield of this. Attempting to create a better learning agent is thus in the field of Computer Science and relevant to my major.

3 Methodology

The methodology is rather simple. First we create a basic framework that allows us to play a simple card game. We made what ended up being a system with discretized turns embedded in a (modified-)finite state machine for the rules of the

game. The game specifically has few rules: on the first turn, one or both players shuffle. On the second turn, one or both players deal. On the third turn and up, the players start taking turns playing a single card, until one is out causing them to win. If somebody takes an incorrect action, they must rectify their mistake by either discarding or drawing a card – depending on whether their mistake caused them to illegally lose or gain a card, respectively.

Next we implement the Cumulus algorithm (Stracuzzi, 2006) and invent the predicates required to learn the game. We evaluate the cumulus algorithm works primarily by hand. That is, I work out the mathematics with the same examples fed into Cumulus and make sure the answer the program gets it close to it (since the data is shuffled I can't get an exact number).

This leads directly into the transfer phase, where we come up with a modification to Cumulus that allows us to apply a neural net to a new, but mostly similar scenario to improve learning rate. Unfortunately, we did not get to this phase, of the experiment, but it ultimately would have been evaluated by how many fewer samples it took to learn the new game with the transfer system vs learning it ground up.

4 Literature Review

In Transfer in Learning by Doing (Krueger, Oates, et al), the paper uses an HMM to bias new scenarios towards old scenarios. It involves learning a game called squawk and simulating a child's learning with a machine. Overall, it concludes that while the second instance of learning a similar game is still very long, by the third game learning time with this method has been decreased monumentally.

In Experimental State Splitting for Transfer Learning (Morrison, Chang, Cohen, Moody), they explore a method for real-time exploration and goal-oriented behavior in ISIS (as a young human would do). The algorithm in question will split states to learn specialized actions based on performance and feedback, and use this knowledge in further scenarios. Overall the method performs better than a non-transfer scheme, except in cases where the agent learns nothing useful (as it to be expected).

In Many Layered Learning (Utgoff, Stracuzzi), we explore the STL algorithm. The STL algorithm is meant to learn labeled, supervised predicates and, when they are learned, use them as a foundation for further predicates. Overall, the algorithm was difficult to interpret and not used as-is by us.

Finally, we have Scalable Knowledge Acquistition Through Cumulative Learning and Memory Organization (Stracuzzi), which illustrates the Cumulus algorithm (among others). Cumulus is in many ways a better STL, and as mentioned is what we used to form out neural net that we were going to use the transfer scheme of. Like STL, what it did was lay a labeled foundation and learned predicates, which is followed up by learning more complex predicates based on these simpler ones – with a catch case to relearn predicates if they don't seem to be learned sufficiently.

5 Results

While we didn't get to the transfer stage, Cumulus seemed to work decently as a learning algorithm. Test cases verified that in just a few games, the neural net was robust enough to successfully recognize the outcomes of many simplified cases.

Further, hand verification for short games confirmed that the math was correct.

There was one notable problem with Cumulus, and that was that it has no sense of state. This was fixed by giving the agent a "memory" and using the memory as an input/output device of the neural net generated by Cumulus. That is, certain neurons would increment certain internal variables, and these variables in turn would be used as the basis for other nodes. Specifically, when an action happened that required a player to play more cards, an internal "how many cards left" variable would be trained to increment, and the agent's node that it is okay to play would only fire in cases where that number was greater than zero.

6 Discussion and Analysis of Project

While, unfortunately, we didn't get to the transfer stage, I did learn about learning algorithms and many transfer schemes. I implemented a complete working game framework for a simple discrete-turn card game and learned that game with Cumulus.

What is prudent, however, is to discuss how to proceed with the transfer learning. While we could have modified one of the items in the literature, we had a few ideas for how to proceed. One problem with Cumulus was that it was labeled, but we managed to work this into an apprenticeship scheme. The idea is that you assume that all neural nets are potentially valid. As the new game is played, eventually your prediction error for each net will go up. After it reaches a given threshold, that net will be discarded until only one remains. For that net we do similarly, except with individual nodes.

As nodes are violated their error goes up, after a certain threshold they are split.

We have to split more fundamental predicates before more complex ones, since the error in a complex, higher layer node may be solely due to error in a lower one. This new node will ask questions of the superviser that will be answered every turn, such as "is this turn an example of this being true or false?" or "give an example of this being true." After this predicate is trained up, we assume it is a valid neural net again, and repeat the process of evaluating error, splitting a node, and asking questions about the new foundation until it is learned.

Another idea was to learn a new neural net from the ground up, but if another pre-learned net seems relevant (i.e. little to no error), to flood the new network with examples generated by the old network to speed up learning without having to go through as many games.

Overall, it is unfortunate we couldn't do the transfer scheme, the first one seems to be interesting, if possibly fallacious to explore, while the second one would likely get decent results. However, this does lay a foundation to possibly explore transfer learning concepts in the future with this framework.

References

[1] Utgoff, Paul; Stracuzzi, David, *Many Layered Learning*.

[2] Krueger, Bill; Oates, Tim; Armstrong, Tom; Cohen, Paul; Beal, Carol, *Transfer in Learning by Doing*.

[3] Morrison, Clayton; Chang, Yu-Han; Cohen, Paul; Moody, Joshua, *Experimental State Splitting for Transfer Learning*.

[4] Stracuzzi, David, *Scalable Knowledge Acquistition Through Cumulative Learning and Memory Organization.*