Brown University
Spring 2019

Weakly-Supervised Classifier Learning via Temporal Logic

Author: Leonard Gleyzer, ScB/Honors Candidate

Advisor: Dr. Michael Littman, PhD

Reader: Dr. Caroline Klivans, PhD

Support: Graduate student Lucas Lehnert

Undergraduate Honors Thesis

Concentration: Computer Science

BROWN UNIVERSITY DEPARTMENT OF COMPUTER SCIENCE

Contents

1 Abstract

One of the primary goals of supervised machine learning is to learn a classifier that can optimally assign a label to unseen instances of data. In fields such as robotics, one would like to classify environment states into semantic categories to support high level task planning based on temporal sensory data. This paper proposes a new methodology to learn a classifier based on temporal sequence data and respective task specifications that these sequences satisfy, using the task-specification language of GLTL, a variant of LTL (Linear Temporal Logic). We show how this formulation can be used to classify handwritten digits from the MNIST database.

2 Introduction

Suppose we have a robot, and we show the robot various paths through an environment (a house, for instance). Ideally, we would like the robot to be able to, from this temporal sensory path information, classify which room is the kitchen, which is the bedroom, etc. We'd like to be able to communicate with the robot in terms that make sense to us—to be able to talk about things like being "in the kitchen". That requires a classifier that can take information about the environment and assess whether such high level properties are true. It is easy to create such a classifier if we have lots of labeled examples; however, we don't always have access to such data. Here, we look at another way these classifiers can be trained. We can consider each room in this scenario as its own atomic proposition (AP), a statement that must be either true or false. For example, consider the phrase "go to the bedroom and then go to the kitchen". From this statement, there are 2 APs we would like to ground: kitchen and bedroom. But how do we communicate this phrase to the robot? There has been a considerable amount of work done in inferring task specifications from demonstrations, but here, rather than inferring task specifications *from* demonstration data, we would like to use task specifications *along with* demonstration data to allow an agent to infer information about the environment over which the demonstration occurred.

This paper is an extension of a research idea proposed by Professor Michael Littman and graduate student Lucas Lehnert. I primarily contributed to this project by implementing the particular neural network algorithms in TensorFlow, as well as ran experiments and collected data to understand

the scope and limitations of the ideas proposed.

3 Related Work

There has been a significant amount of work in the field of reinforcement learning (RL) related to LTL. One such concept is Inverse LTL, of which there have been some interesting developments recently[5][6][7]. The goal of Inverse LTL is to infer an LTL specification that an agent must have satisfied given its actions, given behavioral observations of the agent. This idea is a combination of Inverse RL [3], which attempts to infer a reward function from agent behavior, and LTL, a task-specification language [1] that can be used to define and motivate agent behavior. For instance, Bacchus et al.[2] proposes using LTL-like representations to allow rewards for RL agents to be dependent on history. Littman et al.[1] proposes using LTL specifications directly as a replacement for rewards by having agents select actions to maximize the probability that a given formula is true. One feature of LTL that poses problems is that of infinite traces, that is, a statement such as "eventually be in the kitchen" can have a corresponding sequence of actions of arbitrary length, as long as the last state is "in the kitchen". In this work, the challenge of LTL being defined over infinite traces is met by having temporal operators expire much as temporal discounting can be viewed as rewards expiring.

4 Relevant Background

In order to specify tasks, we need a language for doing so. In this paper, we use Geometric Linear Temporal Logic (GLTL) for this purpose. From GLTL, we can extract Probabilistic Finite State Automata (PFSA), which allow us to express GLTL in a way that can be understood mathematically by our classifier. We define these concepts next.

4.1 Probabilistic Finite State Automaton (PFSA)

A finite-state automaton (FSA) [4] is a theoretical model for accepting/rejecting strings of a given language, where a language is a subset of sequences from a given alphabet of symbols. Generally speaking, a FSA consists of a set of states and rules for transitioning between states based on input received. In a deterministic finite automaton, the transition function is deterministic;

that is, from a given state, each input will cause a deterministic transition. In a probabilistic finite state automaton [8], the transition function is stochastic; that is, from a given state, each input induces a probability distribution over the available states to which the automaton can transition.

Formally a PFSA is defined as a 5-tuple $\langle \Omega, \Sigma, \Delta, \omega_0, \omega_F \rangle$ consisting of

1. a finite set of states Ω,
2. a finite alphabet of input symbols Σ,
3. a transition function $\Delta : \Omega \times \Sigma \times \Omega \to [0, 1]$ and $\forall \omega, \sigma \ \sum_{\omega'} \Delta(\omega, \sigma, \omega') = 1$,
4. an initial start state $\omega_0 \in \Omega$, and
5. an accept state set $\omega_F \subseteq \Omega$ of accept states; if there is a single accept state, we often abuse notation and write $\omega_F \in \Omega$ as the accept state itself.

4.2 Geometric Linear Temporal Logic (GLTL)

Geometric Linear Temporal Logic [1] is a variation of Linear Temporal Logic (LTL). LTL is a logic that captures temporal information [9][10], and whose formulas can be used to encode temporal task specifications. The components of LTL include the usual logical connectives (i.e. negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), logical implication ($\rightarrow$)), as well as temporal modal operators (next ($\bigcirc$), always ($\Box$), eventually ($\Diamond$), until ($\mathcal{U}$)).

Suppose we want to specify to eventually achieve a state g. This can be expressed as $\Diamond g$. If we want to avoid some bad state b, we can say $\Box \neg b$. In the case of a robot going through a house, we can express "go to the kitchen and then go to the bedroom" as $\Diamond(K \wedge \Diamond B)$ (where K is kitchen and B is bedroom), which expresses the statement "eventually have it be the case that we are at the kitchen, from which point we will eventually end up in the bedroom".

GLTL is a variation of LTL that adds an expiration probability ϵ to a formula. For example, let $\epsilon \in (0, 1)$. The formula $\Diamond_\epsilon g$ says that, at each time step, the formula component $\Diamond$ holds with probability $1 - \epsilon$. In effect, this makes the statement say "g eventually holds within t time steps", where $t \sim \mathit{Geometric}(1-\epsilon)$. The feature of GLTL that makes it particularly useful

in this paper is that GLTL formulas can be converted into PFSAs, which are used in the objective function for our optimizer (discussed in Section 5).

For a more in-depth discussion of GLTL, we refer the reader to Littman et al.[1].

4.3 GLTL-PFSA Correspondence Example

As an example of a conversion from a GLTL formula to a PFSA, consider the GLTL formula $\Diamond_{0.01}(a \wedge \Diamond_{0.01} b)$. This formula states we would like to, at some point, get to a, and once we get to a, end in b. The corresponding PFSA (generated by code provided by Cambridge Yang, a graduate student at MIT) is visualized below.

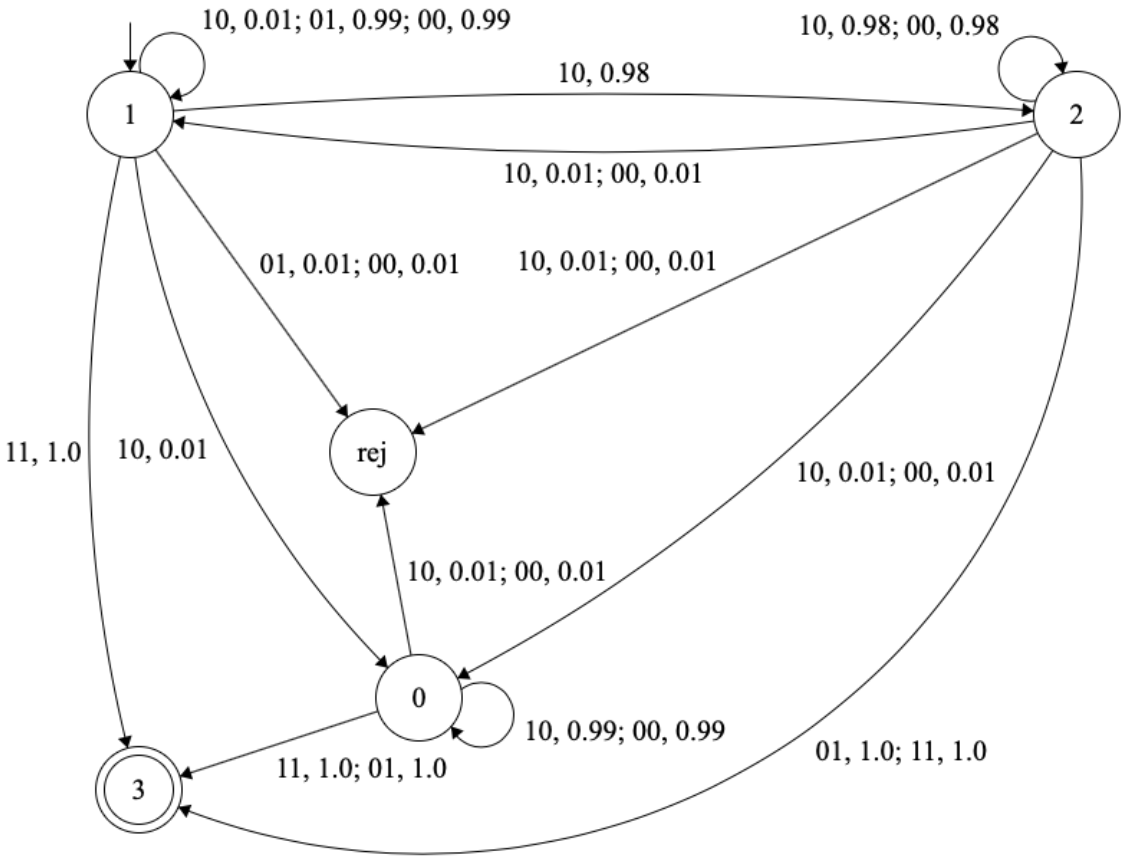

Figure 1: Automaton representing $\Diamond_{0.01}(a \wedge \Diamond_{0.01} b)$

Each edge has a corresponding symbol (some edges have multiple symbols separated by semicolons in the figure for readability), along with a probability of transitioning along that edge given the symbol. Our atomic propositions are $\{a, b\}$, and our symbols are the binary sequences{11, 10, 01, 00}, where 00 means neither a nor b, 10 means only a, 01 means only b, and 11 means both a and b simultaneously. Starting in State 1, until we encounter the first a (symbol 10), we will, with probability 0.99, stay in State 1. Once we encounter the first a, we will, with probability 0.98, transition to State 2 until we encounter the final b (symbol 01), which will cause the transition to State 3, the accept state, with probability 1. All other low probabilities and states are related to the temporal operators probabilistically expiring.

5 Problem Formulation

In the traditional supervised learning setting, we are given a training set (X, Y), where X is the set of training data, and Y is the corresponding set of training labels. In our scenario, we can think of each element of X as a finite sequence (which we will call a *trajectory*), and the corresponding element of Y a GLTL formula that the respective trajectory satisfies. For example, suppose we have the MNIST handwritten digit database. An example of an element $(x, y) \in (X, Y)$ is $([\mathbf{00101211210}], \Diamond_{0.01}(2 \wedge \Diamond_{0.01} 0))$, where **2** is an MNIST instance of the digit 2, and so forth.

Given a trajectory τ over a dataset D of states S and corresponding GLTL formula G, we can convert G to a corresponding PFSA

$$A_G = \langle \Omega_G, \Sigma_G, \Delta_G, \omega_G^0, \omega_G^F \rangle.$$

Here, Σ_G consists of all true/false (`T/F`) assignments of propositions in G. For example, if $G = \Diamond_{0.01}(2 \wedge \Diamond_{0.01} 0)$, then $\Sigma_G = \{$`FF, FT, TF, TT`$\}$, with respect to 0 and 2.

We would like to find a function f (called a grounding function) over S that assigns a probability distribution to each $s \in S$ such that the highest probability is assigned to the symbol that represents the ground truth of what s represents. For instance, in the kitchen-bedroom example, we want f to take in sensory data from being in the kitchen, and output a probability distribution over the available symbols such that the highest probability (ideally as close to 1 as possible) is assigned to the symbol representing kitchen=`T` and bedroom=`F`.

Here, f is parameterized by some latent parameters $\boldsymbol{\theta}$. In our case, $\boldsymbol{\theta}$ is the weights and biases of a neural network that, given a trajectory, outputs a matrix where each row is a discrete probability distribution over symbols corresponding to each state in the trajectory. We would like to find an optimal $\boldsymbol{\theta}^*$ such that, for a dataset D of N trajectory-automaton pairs (τ_i, A_{G_i}), the probability of transitioning from $\omega^0_{G_i}$ to $\omega^F_{G_i}$ over τ_i given $\boldsymbol{\theta}^*$, for all i, is maximized, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} \mathbb{P}\left[\omega^0_{G_i} \xrightarrow{\tau_i} \omega^F_{G_i} \middle| \boldsymbol{\theta}\right]. \tag{1}$$

Because we can explicitly tabulate Ω and Σ, we can use Δ_G to construct state-to-state transition matrices for each $\sigma \in \Sigma$

$$\Delta^\sigma_G \doteq [\Delta^\sigma_G(i,j)] = [\Delta_G(i,\sigma,j)]. \tag{2}$$

For a trajectory τ of length T, for each step t, we can compute, from $f_{\boldsymbol{\theta}}(s_t, \sigma)$ (where we write $f_{\boldsymbol{\theta}}(s,\sigma)$ to mean the entry of the probability distribution $f_{\boldsymbol{\theta}}(s)$ corresponding to the symbol σ), the expected state-to-state transition matrix

$$\Delta^t_G \doteq \mathbb{E}_{\sigma_t}[\Delta^{\sigma_t}_G] = \sum_{\sigma} \Delta^\sigma_G \cdot f_{\boldsymbol{\theta}}(s_t, \sigma). \tag{3}$$

Assuming that transitions are Markov, starting from ω^0_G and ending at ω^F_G, we have

$$\mathbb{P}\left[\omega^0_G \xrightarrow{\tau} \omega^F_G \middle| \boldsymbol{\theta}\right] = \boldsymbol{e}^\top_{\omega^0_G} \left(\prod_{t=1}^{T} \Delta^t_G\right) \boldsymbol{e}_{\omega^F_G}. \tag{4}$$

In the case of our neural network, $f_{\boldsymbol{\theta}}$ takes in a state vector $\boldsymbol{s}$, and outputs the result of applying the *softmax* function to the vector output of passing $\boldsymbol{s}$ through a neural network parameterized by $\boldsymbol{\theta}$, the weights and biases of the network.

By using this method, we expect to obtain a locally optimal $\boldsymbol{\theta}^*$ by computing gradients $\nabla_{\boldsymbol{\theta}} f$.

Ultimately, the objective function we want to maximize is the per-trajectory probability of transitioning from the automaton start state to the automaton accept state via the given trajectory. Internally, for each state in a given

trajectory, there is a discrete probability distribution over which automaton state we are currently in. As we train on more and more trajectories, the probabilities relating to which automaton states we transition through become more deterministic. As this happens, our model is able to better classify which symbols (classification) to assign states it encounters, which in turn configures the automaton state probabilities to better reflect the ground truth of what is happening.

6 Experimental Results

In the following sections, we summarize the main experimental findings and offer corresponding explanations and discussions.

6.1 Gridworld Trials

We first started with a simple scenario to evaluate the validity of our objective function. We had sixteen states arranged in a 4×4 grid, enumerated below.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

We wanted to give the network specific trajectories and GLTL formulas that would cause it to associate the bottom-left square (state 12) with the color green, the bottom-right square (state 15) with the color blue, and the remaining squares with no color. We gave the following 3 trajectory–GLTL formula pairs:

$$([\mathbf{13}, \mathbf{14}, \mathbf{15}], \Box_{0.01} \neg g \wedge \Diamond_{0.01} b)$$
$$([\mathbf{14}, \mathbf{13}, \mathbf{12}], \Box_{0.01} \neg b \wedge \Diamond_{0.01} g)$$
$$([\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{7}, \mathbf{6}, \mathbf{5}, \mathbf{4}, \mathbf{8}, \mathbf{9}, \mathbf{10}, \mathbf{11}], \Box_{0.01} \neg (g \vee b))$$

where a bold number is presented to the learner as the one-hot vector encoding that number. A one-hot vector is a vector used to encode categorical data that has a 1 in only one position and 0s in all others. For example, the state "12" would be encoded as a 16-bit vector with a 1 in position 12 and 0s everywhere else. After training on just these three points over 25 epochs

(which took roughly 3 seconds), the probability distributions are almost deterministic. The visualization of the learned grid is shown below.

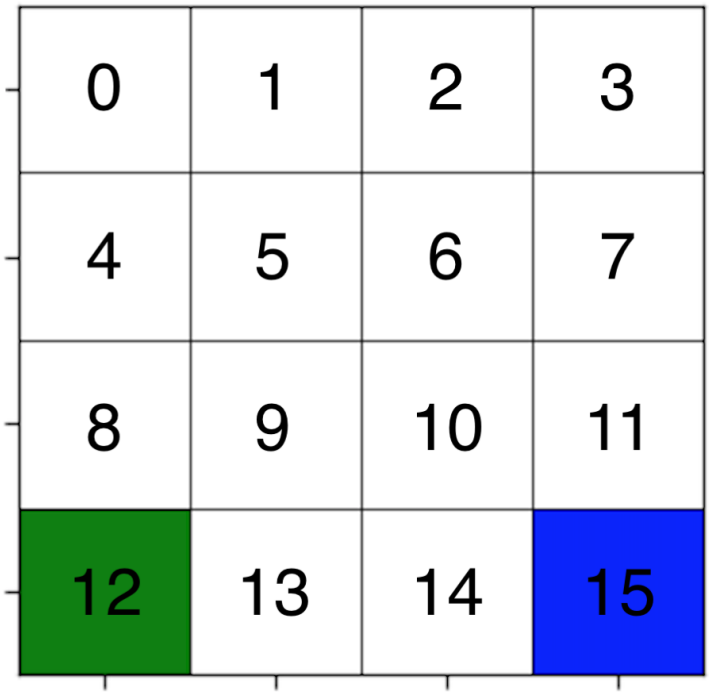

Figure 2: Gridworld Learned Colors

We can see that every square is properly labeled with the color symbol we had in mind when choosing the training data. Although this example worked well, one example that did not work as hoped was the formula $\Diamond_{0.01}(\Box_{0.01} g)$ along with a trajectory that went through states 0, 1, 5, and 6, and then circled around states 10, 11, 14, and 15 for a while. The idea of this example was to see if the learner would recognize that our formula specified "eventually always green", that is, at some point, always be in the green. We wanted for the learner, training on this trajectory, to recognize the bottom-right quadrant (states 10, 11, 14, and 15) as green, and the other states it touched as colorless. However, the learner converged to labeling all states it had touched as green, not only the quadrant we had intended. An ex-

planation for this observation is that the labeling the learned proposed led to a high objective function value, and was perhaps easier to arrive at than the labeling we had imagined. Furthermore, states here were represented as one-hot vectors, and we should expect vectors encoding perceptual (e.g. visual) information to be different, since no two instances of perceptual information (such as two distinct images) will be exactly identical, unlike the one-hot vectors used in this example.

6.2 MNIST Trial 1: Two Temporal Operators

With these observations in mind, we moved onto a more challenging problem. Can we classify handwritten digits using this method? The formulation of the problem is nearly identical. Instead of passing in one-hot vectors, we pass in vectors representing handwritten digits from the MNIST database, examples from which are shown below.

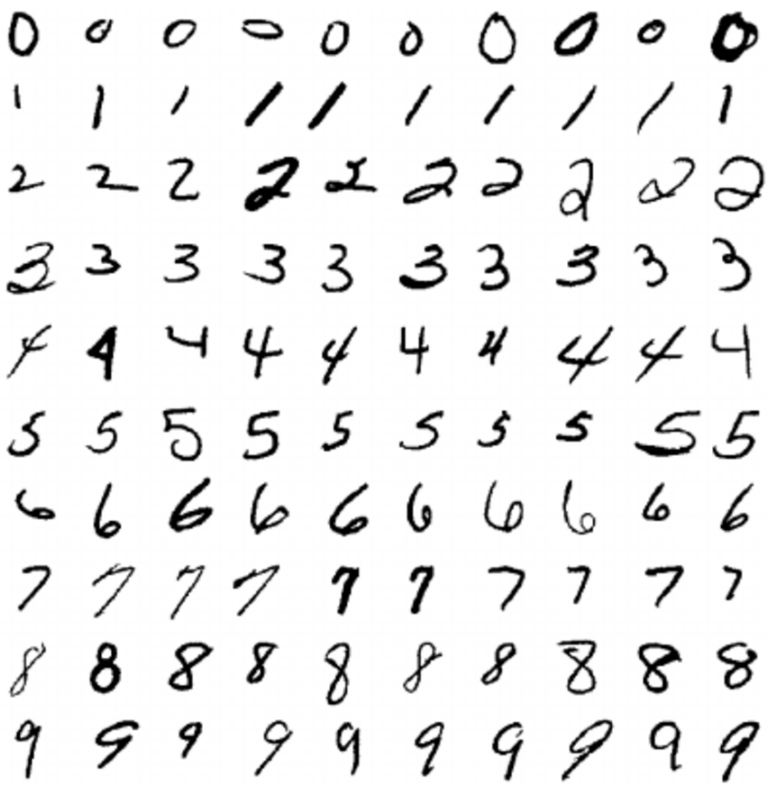

Figure 3: MNIST digits, source: Hiromichi Fujisawa, ResearchGate.net

Each digit is represented as a 28×28 array, where each entry is the 0-255 grayscale value of the pixel in that position. We normalize all values by dividing all entries by 255, so values are constrained to the unit interval [0, 1], and reshape each 28×28 array to a single-dimension 784-element vector. During training, we pass in one trajectory at a time, and, after each output, Equation (4) is used to calculate loss (or rather the negation of Equation (4), since the optimizers in TensorFlow only do minimization). The training process is the same as in the Gridworld example, except we pass in 784-length MNIST vectors rather than 16-length one-hot vectors.

Using the digits 0, 1, and 2, we wanted to test whether images can be recognized after training using formulas with multiple temporal operators such as $\Diamond_{0.01}(2 \wedge \Diamond_{0.01} 0)$, which effectively says "end up in 0, but only after you reach a 2".

To generate a trajectory that satisfies this formula, we set 20 as our trajectory length. The first half of the elements were randomly selected MNIST digits over Uniform{0, 1}, the second half were randomly selected MNIST digits over Uniform{1, 2}, and a randomly-selected 0 was appended to the end.

Using only the two formulas $\Diamond_{0.01}(2 \wedge \Diamond_{0.01} 0)$ and $\Diamond_{0.01}(0 \wedge \Diamond_{0.01} 2)$, we trained the model on trajectories generated in the manner described above. After training, we generated a test set consisting of 900 0s, 900 1s, and 900 2s that the network had not previously encountered (single digits are trajectories of length 1). On this test set, our trained model was able to achieve 97% classification accuracy.

This result signified that not only was the network able to recognize the 0s and 2s that were present in the formula, but it was also able to properly label 1s, so the logic was in a way propagated through the unseen symbols.

6.3 MNIST Trial 2: No Explicit Digit Location "Eventually" Operator Only

Next, we wanted to test whether we could learn to classify digits when none of the formulas explicitly state that a single digit will always be a certain position, unlike in the previous example, where $\Diamond_{0.01}(2 \wedge \Diamond_{0.01} 0)$ signified that 0 (and only 0) should always be the last element in a trajectory.

Using the digits 0–3, we used the following formulas.

$$\Diamond_{0.01}(0 \vee 1)$$
$$\Diamond_{0.01}(1 \vee 2)$$
$$\Diamond_{0.01}(2 \vee 3)$$
$$\Diamond_{0.01}(3 \vee 0)$$

This set of formulas ensures that no single digit is always to be expected at a particular trajectory position. A trajectory corresponding to $\Diamond_{0.01}(0 \vee 1)$, for instance, would have the first 19 elements sampled from Uniform$\{2, 3\}$ and the last element randomly sampled from Uniform$\{0, 1\}$.

After training on these formula–trajectory pairs, we achieved a classification accuracy of 97% on a test set consisting of 900 (each) of 0s, 1s, 2s, and 3s that were not previously encountered.

This result shows that the network was able to differentiate symbols even though each formula never singled out a single digit as always being in a particular position in the trajectory, as we had intended.

6.4 MNIST Trial 3: No Explicit Digit Location "Always" Operator Only

However, not all formulas led to good classification. For example, we tested the following set of formulas with the digits 0–3:

$$\Box_{0.01}(0 \vee 1)$$
$$\Box_{0.01}(1 \vee 2)$$
$$\Box_{0.01}(2 \vee 3)$$
$$\Box_{0.01}(3 \vee 0)$$

These formulas state that every element of the trajectory is one of the two digits specified in the formula.

Trajectories were created from a uniform distribution over the 2 digits specified by a formula. The result of testing this set of formulas resulted in all test digits being classified with the same symbol as multiple or sometimes even all digits 0–3 simultaneously. The subset of digits in the labels varied among runs. Sometimes it labeled all digits as "simultaneously 0, 1, 2"; sometimes it was "simultaneously 1, 2, 3"; sometimes it was "simultaneously 0, 1, 2,

3". This most likely depended on which random images were sampled. In any case, all test digits were always assigned the same symbol every time.

An explanation for this result is that, in the case where we only used "eventually" formulas, a single trajectory encoded both positive and negative examples. For instance in a length 20 trajectory encoding "eventually 0 or 1", the first 19 digits are negatives examples (not 0 or 1), and the last digit is a positive example (0 or 1). In the case where we use only "always" formulas, we only have positive examples. This allows for less differentiation among digits per trajectory, and as a result, the classifier finds that the easiest way to satisfy all formulas is to simply assign each digit the symbol/label of being multiple digits simultaneously, which technically does satisfy the given formulas.

6.5 MNIST Trial 4: No Explicit Digit Location, "Eventually" and "Always" Operators Together

When testing both sets of formulas together, that is,

$$\Diamond_{0.01}(0 \vee 1)$$
$$\Diamond_{0.01}(1 \vee 2)$$
$$\Diamond_{0.01}(2 \vee 3)$$
$$\Diamond_{0.01}(3 \vee 0)$$
$$\Box_{0.01}(0 \vee 1)$$
$$\Box_{0.01}(1 \vee 2)$$
$$\Box_{0.01}(2 \vee 3)$$
$$\Box_{0.01}(3 \vee 0)$$

we were able to attain 97% accuracy, and at a faster rate than when only using "eventually" examples. An explanation for this is that the "eventually" examples provide both positive and negative instances with good accuracy on their own, and the added "always" examples add more positive examples and differentiation to achieve the desired result even faster. Number of trajectories vs. Accuracy obtained on test set for the above three training scenarios is plotted below.

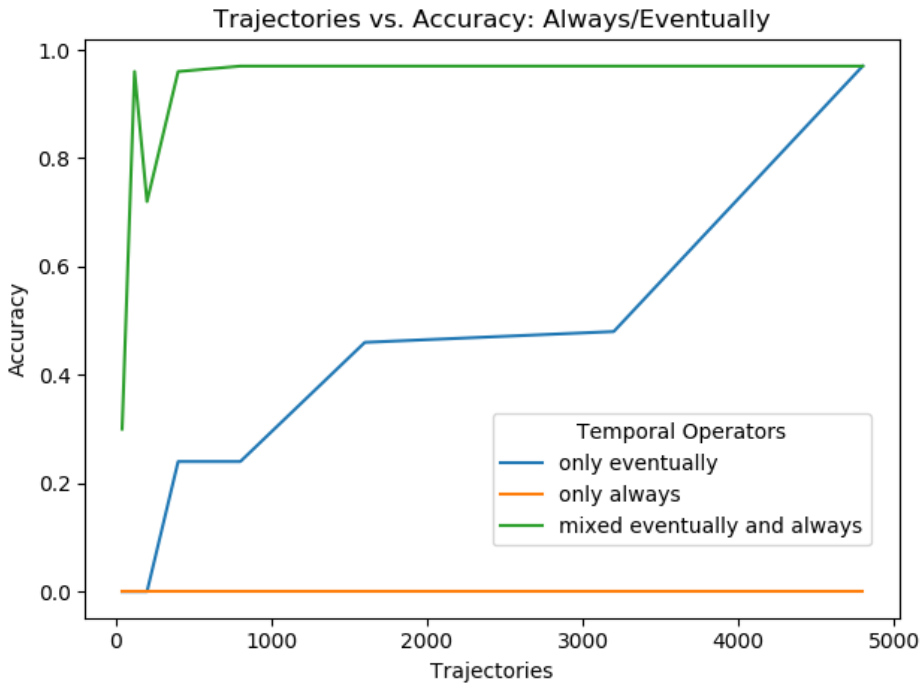

Figure 4: Trajectories vs. Accuracy: Always/Eventually

Each "always" trajectory had length 30, and each "eventually" trajectory had length 20. There was an equal number of trajectories for each of the eight formulas listed above. We can see that after around 400 trajectories, the mixed always/eventually has reached a stable point with high accuracy. However, it took over 4000 trajectories training using only "eventually" operators to reach the same accuracy. Using only "always" operators never produces above 0 accuracy.

Another instance where classification is incorrect is if instead of testing (with digits 0-2) both $\Diamond_{0.01}(2 \wedge \Diamond_{0.01}0)$ and $\Diamond_{0.01}(0 \wedge \Diamond_{0.01}2)$ together, we only test one of them. This leads to the classifier assigning all test digits a label of either 1 or simultaneously 1 and the digit actually represented. A possible explanation is that variations of the digit 1 are found in both halves of the given trajectories, and because no two digits are identical, the classifier is unable to let the desired logic propagate. For instance, if we test $\Diamond_{0.01}(2 \wedge \Diamond_{0.01}0)$ with one-hot vectors representing digits rather than MNIST vectors, in which case there is no variation among digits of the same kind,

the desired logic does propagate through, and we achieve perfect classification on one-hot vectors. By desired logic, we mean the following: take, for instance, the one-hot trajectory [**10110211210**]. If given the formula $\Diamond_{0.01}(2 \wedge \Diamond_{0.01}0)$, the classifier should, in theory, recognize the last entry as 0, which means that the other one-hot 0s will also have the proper classification. Furthermore, we should only get to a terminal 0 after getting the 2 first, and because there are 0s after 1s in the beginning, the 1s should not be classified as 2s, and we already know they are not 0s because we have already recognized zeros, which allows for proper differentiation and classification of all 3 digits.

6.6 Internal PFSA Probabilities

Consider the formula/automaton described in Section 4.3, except with a and b replaced with 0 and 1, respectively, so that the formula reads $\Diamond_{0.01}(0 \wedge \Diamond_{0.01}1)$ (the automaton remains unchanged). In the plots on the following page, we summarize the results of running the stated formula with trajectories of the form [**11111000001**]. The plots below (from every 50 trajectories) show the progression of how internal automaton probabilities (y-axis) evolve through the trajectory states (x-axis) as the training increases. State 1 represents the state of "not yet encountered the first 0". State 2 represents the state of "encountered the first 0 but not yet the final 1". States 0 and 4 corresponds to low-probability states corresponding to GLTL formula expirations. State 3 is the accept state, representing the state of "encountered the final 1". State 5 is an additional state we added in experimentation for logistic reasons (essentially, it is a state representing full completion of a pass through the automaton; without this additional state, the code would not function properly); we include it in the plots below to make the probability distributions complete. The legend is shown below.

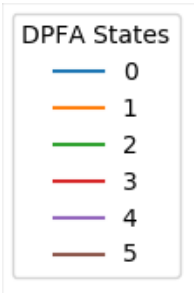

Figure 5: Legend for PFSA States

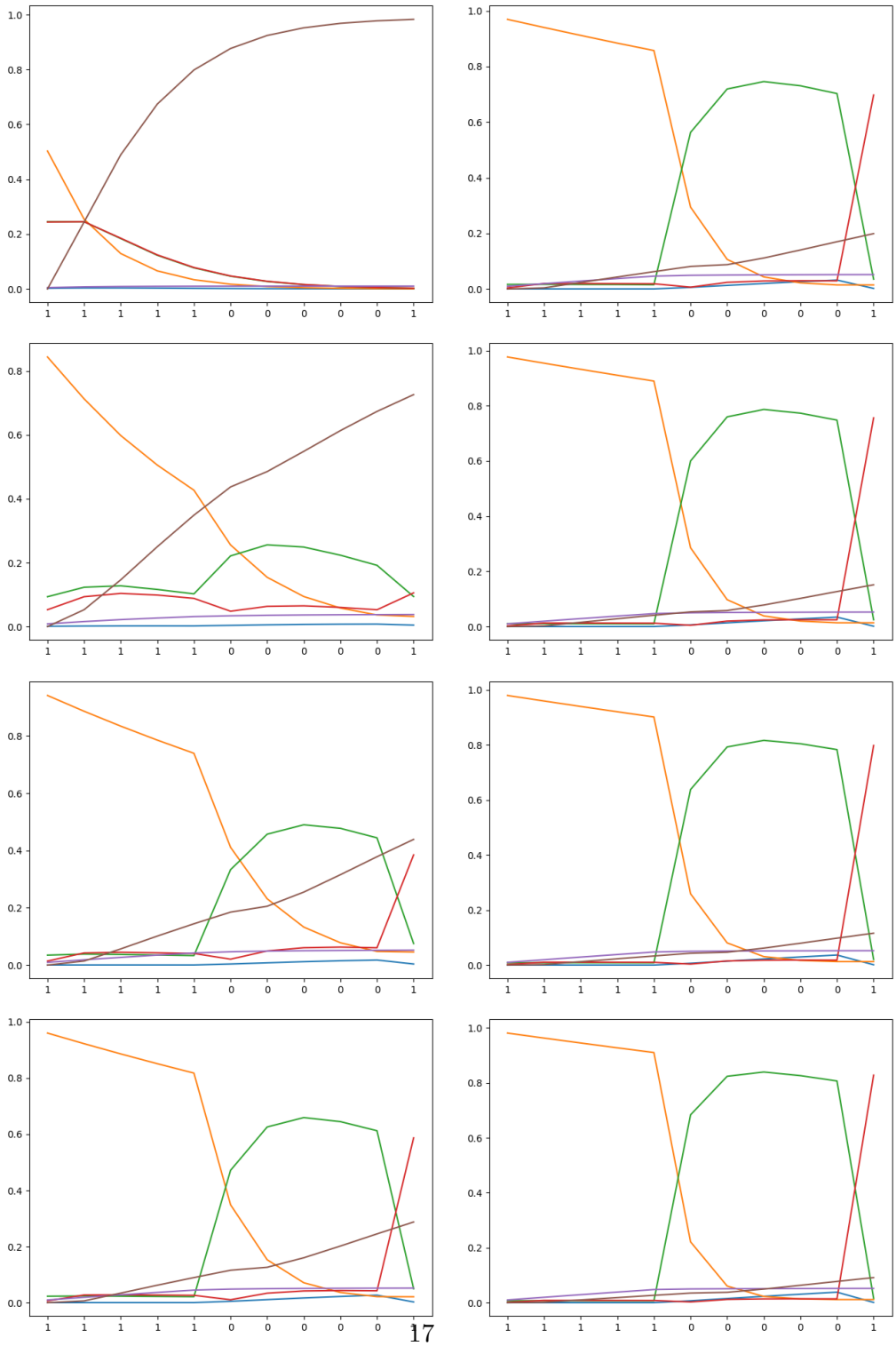

Figure 6: Evolution of Internal PFSA Probabilities

We can see that, when training starts, the probabilities are either not clear or are concentrated on the wrong states as the first trajectory is processed. As training continues, we can see that the internal probabilities are configured so that paths through the automaton more accurately reflect the trajectory passed in. Looking at the last plot, the first 5 trajectory elements (which are 1s) have corresponding automaton probabilities concentrated on state 1, which represents the state of "not yet encountered the first 0", and stays there as a result of receiving (correctly) the symbol (zero=`F`, one=`T`). The next state marks a transition to automaton state 2, which results from the recognition of the first zero, with symbol (zero=`T`, 1=`F`). The automaton has highest probability in this state for 5 consecutive trajectory elements (which are indeed all 0s). The last trajectory state marks the final transition to state 3, which is the accept state, i.e. final state of the automaton path representing having seen the ending "one" after the first "zero"; the probability is indeed concentrated on this automaton state in the last row of the table. We can also see that the majority of time in this case is spent on fine-tuning the probabilities, as the coarse configuration can be seen after the first 150-200 trajectories.

7 Conclusion

From the experimental results, we can see that our network is able to perform simple image classification using temporal image-sequence data and corresponding GLTL-derived automata. This approach shows promise for more complex tasks.

8 Future Work

One main drawback of the current implementation is that, because the PFSA created from the GLTL assigns true/false to all propositions, the number of symbols is exponential in the number of propositions, making the problem intractable for classification problems with many possible class labels. If we implement the GLTL-to-automaton translation in a different way that would make the number of symbols polynomial in the number of atomic propositions, the problem would become much more tractable.

Other future directions include testing this idea on more complex datasets, such as robotics image data of movement among various rooms in some environment. Another idea is training a drone to recognize components of the

landscape it flies over by training it on images of flight trajectories with corresponding GLTL formulas representing sentiments such as "this path has two trees" or "this path ends in a person who is injured and needs help".

Another direction is to increase the complexity of the network. The network used for this paper was a simple feed-forward network with one hidden layer. Figuring out how to add a convolutional layer to work with trajectory data may offer performance boosts.

Other potential ideas include testing this on non-image data, such as audio data or gene sequence data.

9 Acknowledgements

I would like to thank Professor Michael Littman for advising my thesis, Professor Caroline Klivans for being my thesis reader, Lucas Lehnert for formulating (along with Professor Littman) the initial theory for this project, as well as frequently collaborating and providing much help and support, Kevin Du for providing necessary support code for running experiments, Cambridge Yang at MIT for providing the code for the GLTL converter, and everyone else who helped collaborate on/contribute to this project in any way.

References

[1] Littman, Michael L., Topku, Ufuk, Fu, Jie, Isbell, Charles, Wen, Min, and MacGlashan James. *Environment-Independent Task Specifications via GLTL*. ArXiv, 2017.

[2] Bacchus, Fahiem, Boutilier, Craig, and Grove, Adam. *Rewarding Behaviors*. AAAI'96 Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2, 1996.

[3] Ng, Andrew Y. and Stuart, Russell *Algorithms for Inverse Reinforcement Learning*. ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning pages 663-670, 2000.

[4] Sipser, Michael *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[5] Shah, Ankit, Kamath, Pritish, Li, Shen, and Shah, Julie *Bayesian Inference of Temporal Task Specifications from Demonstrations*. 32nd Conference on Neural Information Processing Systems, 2018.

[6] Vazquez-Chanlatte, Marcell, Jha, Susmit, Tiwari, Ashish, Ho, Mark K., and Seshia, Sanjit A. *Learning Task Specifications from Demonstrations*. 32nd Conference on Neural Information Processing Systems, 2018.

[7] Kasenberg, Daniel and Scheutz, Matthias *Interpretable Apprenticeship Learning with Temporal Logic Specifications*. ArXiv, 2017.

[8] Vidal, Enrique and Thollard, Franck, de la Higuera, Colin, Casacuberta, Francisco, and Carrasco, Rafael C. *Probablilistic Finite-State Machines – Part I*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 7. July 2005.

[9] Manna, Zohar and Pnueli, Amir. *The Temporal Logic of Reactive & Concurrent Sys.*. Springer, 1992.

[10] Baier, Christel and Katoen, Joost-Pieter. *Principles of Model Checking*. MIT Press, 2008.