Model Regularization for Stable Sample Rollouts

Erik Talvitie
Department of Mathematics and Computer Science
Franklin & Marshall College
Lancaster, PA 17604

Abstract

When an imperfect model is used to generate sample rollouts, its errors tend to compound – a flawed sample is given as input to the model, which causes more errors, and so on. This presents a barrier to applying rollout-based planning algorithms to learned models. To address this issue, a training methodology called "hallucinated replay" is introduced, which adds samples from the model into the training data, thereby training the model to produce sensible predictions when its own samples are given as input. Capabilities and limitations of this approach are studied empirically. In several examples hallucinated replay allows effective planning with imperfect models while models trained using only real experience fail dramatically.

1 INTRODUCTION

Online Monte Carlo-based planning algorithms such as Sparse Sampling (Kearns et al. 2002), UCT (Kocsis & Szepesvári 2006), and POMCP (Silver & Veness 2010) are attractive in large problems because their computational complexity is independent of the size of the state space. This type of planning has been successful in many domains where a perfect model is available to the agent, but there is a fundamental barrier in the way of applying these methods to *learned* models. The core operation in Monte Carlo planning is sampling possible futures by composing the model's one-step predictions (in a process called *rollout*). When an imperfect model is used, errors in one sampled observation cause more errors in the next sample, and so on until the rollout bears little or no resemblance to any plausible future, making it worthless for planning purposes.

To illustrate, consider the following simple planning problem, which will serve as a running example throughout the paper. In "Mini-Golf" (pictured in Figure 1a), the agent's observations are 20×3 binary images. The image shows a ball, a wall, and a pit. A cover slides back and forth over the pit. The agent selects between two actions: *no-op* and *hit*; once the *hit* action is selected, only the *no-op* action is available thereafter. When the ball is hit, it travels to the right, knocks the wall down (i.e., the wall disappears), bounces back to the left, hits the left edge, and bounces back to the right. If the cover is in the correct position, the ball reaches the right side, the episode ends, and the agent receives 1 reward. Otherwise, the ball falls into the pit which ends the episode with no reward.

Imagine that the agent learns a factored model over pixels. There is a separate component model responsible for predicting each pixel; the prediction for the full image is the product of the component models' predictions. The component model at position $\langle x, y \rangle$ is given the pixel values in an $n \times m$ neighborhood centered at $\langle x, y \rangle$ for the previous two time-steps, as well as the most recent two actions.

If $n = 9$ and $m = 3$, there is enough information to make perfectly accurate predictions, but imagine that $n = 7$. This simulates the common situation that the minimum set of features necessary for perfect accuracy is either unknown or computationally impractical. The consequences of the limitation are illustrated in Figure 1b. The main problem is that a pixel immediately adjacent to the ball cannot tell if the ball is next to a wall, and therefore whether the ball will bounce back. As a result, the model for this pixel is uncertain whether the pixel will be black or white.

Figure 2 shows a rollout from a perfect model ($n = 9$) on the left and a rollout from an imperfect model ($n = 7$) in the center. When the model is very accurate, rollouts are plausible futures. When the model is imperfect, the model's limitation eventually causes an error – a pixel next to the ball stochastically turns black, as if the ball were bouncing back. This creates a context for the nearby pixel models that is unlike any they have seen before. This causes new artifacts to appear in the next sample, and so on. The meaningful structure in the observations is quickly obliterated making it impossible to tell whether the ball will eventually reach the other side.

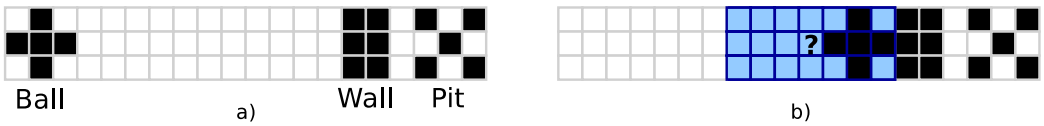

Figure 1: a) The Mini-Golf problem. b) The pixel marked with the ? cannot be predicted perfectly with a 7×3 neighborhood (shaded) because it cannot tell if the ball will bounce off of a wall.

This issue can be framed as a mismatch between the model's training data and test data. Specifically, the model's training data is drawn from real world experience but when it is actually used for planning (i.e., "tested"), the inputs to the model are samples generated from the model itself, which may be very different than real observations. In a sense, the model in this example has overfit to the real world! The approach taken in this paper to address this train/test mismatch is to mix samples from the model into the training data. The resulting method is called "hallucinated replay," in analogy to "experience replay" (Lin 1992). On the right of Figure 2 is a rollout of a model trained using hallucinated replay. Note that it makes the same initial error (a spurious black pixel still appears), but it has learned that lone black pixels should turn white. In effect, it has learned to correct for its own sampling errors.

This paper has two main goals. The first is to highlight the dramatic impact of this mismatch between real inputs and sampled inputs on model-based reinforcement learning. In several examples seemingly innocuous model limitations cause catastrophic planning failures. The second is to empirically investigate hallucinated replay as an approach to addressing this issue. Most notably, hallucinated replay will be used to learn imperfect models that can nevertheless be used for effective planning. These results indicate that this method may be an important tool for model-based reinforcement learning in problems where one cannot expect to learn a perfect model.

2 ENVIRONMENTS AND MODELS

The development of hallucinated replay will focus on the setting of multi-dimensional, discrete, deterministic dynamical systems. Time proceeds in discrete steps. At each step t, the agent selects an action a_t from a finite set of actions $\mathcal{A}$. Given the action, the environment deterministically produces an observation vector $\mathbf{o}_t$. Let $\mathcal{O} = \mathcal{O}_1 \times \mathcal{O}_2 \times \ldots \times \mathcal{O}_n$ be the observation space, where $\mathcal{O}_i$ is the finite set of possible values for component i of the observation. Note that *not all* $\mathbf{o} \in \mathcal{O}$ will necessarily be reachable in the world. Let $\mathcal{O}_E \subseteq \mathcal{O}$ be the subset of observations that the environment can ever produce. For example, in Mini-Golf, $\mathcal{O}$ would be the set of *all* 20×3 binary images, and $\mathcal{O}_E$ would be the set of binary images with a ball, a pit, and either a wall or no wall.

Let $\mathcal{H}$ be the set of all sequences $a_1\mathbf{o}_1 \ldots a_k\mathbf{o}_k$ for all lengths k and let $\mathbf{o}_t = T(a_t, h_{t-1})$ where $h_{t-1} = a_1\mathbf{o}_1 \ldots a_{t-1}\mathbf{o}_{t-1}$ is the *history* at time $t-1$ and T is the *transition function* $T : \mathcal{A} \times \mathcal{H} \to \mathcal{O}_E$. Note that typically not all sequences in $\mathcal{H}$ are reachable. Let $\mathcal{H}_E \subseteq \mathcal{H}$ be the set of histories that could be produced by taking some action sequence in the environment.

At each step the environment also produces a reward value r_t and, in the episodic case, a Boolean termination signal ω_t. For simplicity's sake, these values are assumed to be contained in the observation vector $\mathbf{o}_t$ so correctly predicting the next observation also involves correctly predicting the reward and whether the episode will terminate. Assume that if $\omega_t = True$ then $\omega_{t'} = True$ and $r_{t'} = 0$ for all $t' > t$. The goal of an agent in this environment is to maximize the expected discounted return $\mathrm{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right]$, where $\gamma \in [0, 1]$ is the discount factor and the expectation is over the agent's behavior (the environment is deterministic).

Note that even though the environment is assumed to be deterministic, it may be too complex to be perfectly captured by a model. Thus models will, in general, be stochastic. Also note that because $\mathcal{O}_E$ and $\mathcal{H}_E$ are unknown *a priori*, models will be defined over the full spaces $\mathcal{O}$ and $\mathcal{H}$ instead. So, an imperfect model not only has an incorrect probability distribution over possible observations, but it may assign positive probability to *impossible* observations, as seen in the Mini-Golf example. Furthermore, during rollouts the model will have to make predictions given histories in $\mathcal{H} \backslash \mathcal{H}_E$, though no amount of experience in the environment will provide guidance about what those predictions should be.

In an abstract sense, a *model* M provides a conditional probability $T_M(\mathbf{o} \mid ha)$ for every observation vector $\mathbf{o} \in \mathcal{O}$, action $a \in \mathcal{A}$, and history sequence $h \in \mathcal{H}$. As assumed above, predicting the next observation also involves predicting the reward value and whether the episode will terminate. Note that the history h is an arbitrarily long and ever expanding sequence of actions and observations. Most practical models will rely upon some finite dimensional summary of history $\mathbf{c}(ha)$, which is hereafter called the model's *context*. For the sake of simplifying notation, let $\mathbf{c}_t = \mathbf{c}(h_{t-1}a_t)$. For instance, a Markov model's context would include the most recent action and observation and a POMDP model's context would include the belief state associated with the history.

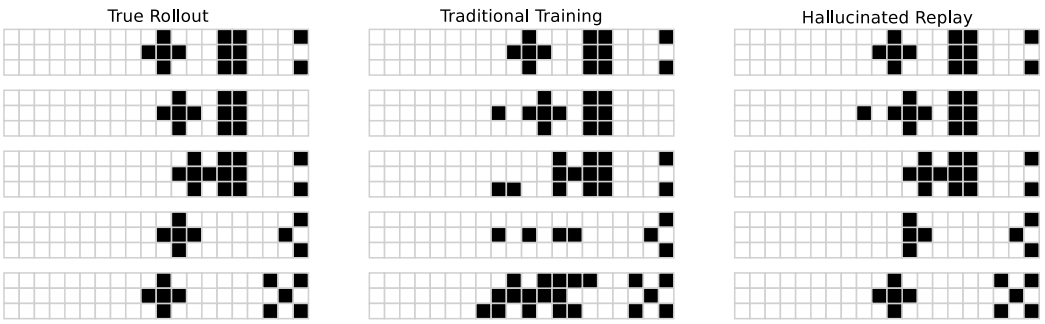

Figure 2: Rollouts using a perfect model (left), a model trained with only real experience (center) and a model trained with hallucinated replay (right). Hallucinated replay makes the model robust to its own errors.

3 HALLUCINATED REPLAY

Consider learning a model using a set of training trajectories $\{h^1, \ldots, h^k\}$, where trajectory h^i is of length $|h^i|$. The training data can be seen as a set of training pairs for a supervised learning algorithm: $\{\langle \mathbf{c}_t^i, \mathbf{o}_t^i \rangle \mid i = 1 \ldots k, t = 1 \ldots |h^i|\}$, where the context $\mathbf{c}_t^i$ is the input and the observation $\mathbf{o}_t^i$ is the target output. Let $\mathcal{M}$ be the space of possible models, or the *model class*. Most model learning methods aim to find the model $M^* \in \mathcal{M}$, which maximizes the log-likelihood:

$$M^* = \operatorname*{argmax}_{M \in \mathcal{M}} \sum_{i=1}^{k} \sum_{t=1}^{|h^i|} \log(T_M(\mathbf{o}_t^i \mid \mathbf{c}(h_{t-1}^i a_t^i)))$$

$$= \operatorname*{argmax}_{M \in \mathcal{M}} \sum_{i=1}^{k} \sum_{t=1}^{|h^i|} \log(T_M(\mathbf{o}_t^i \mid \mathbf{c}_t^i))$$

If a perfect model exists in $\mathcal{M}$, one for which $T_M(\mathbf{o}_t^i \mid \mathbf{c}_t^i) = 1$ for all i and t, then M^* is a perfect model. So in this case M^* is a sensible learning target for the purposes of model-based reinforcement learning.

If there is no perfect model in $\mathcal{M}$, M^* may not be a very good model for planning. Note that the log-likelihood only measures the accuracy of the model's one-step predictions given contexts produced by the environment. However, as seen above, in order to prevent errors from compounding during planning, the model must make sensible predictions *in contexts the environment will never produce*. As will be demonstrated empirically, a model that accomplishes this may yield better planning performance than M^*, even if it incurs more prediction error (lower log-likelihood).

The high-level idea behind *hallucinated replay* is to train the model on contexts it will see during sample rollouts in addition to those from the world. Hallucinated replay augments the training data by randomly selecting a context $\mathbf{c}_t^i$ from the training set, replacing it with a "hallucinated" context $\hat{\mathbf{c}}_t^i$ that contains sampled observations from the model, and finally adding the training pair $\langle \hat{\mathbf{c}}_t^i, \mathbf{o}_t^i \rangle$ to the training set. The model is trained with inputs that are generated from its own predictions, but outputs from real experience.

The experiments below consider three concrete instantiations of this abstract idea, which are described below and illustrated in Figure 3. All three begin by randomly selecting a trajectory i and timestep t to select the "output" component of the new training pair.

Shallow Sample: The simplest form of hallucinated replay is to replace the observation at step $t-1$ with a sample $\hat{\mathbf{o}}_{t-1}^i \sim T_M(\cdot \mid \mathbf{c}_{t-1}^i)$. Then the hallucinated context is $\hat{\mathbf{c}}_t^i = \mathbf{c}(h_{t-2}^i a_{t-1}^i \hat{\mathbf{o}}_{t-1}^i a_t^i)$. This approach essentially imagines that there might be errors in the first sample of a rollout (the sampled observation placed at the end of history) and trains the model to behave reasonably despite those errors (i.e., to predict the true next observation).

Deep Sample: A flaw in the above approach is that, even if a rollout recovers from an error the erroneous observation will still persist in history, possibly affecting predictions deeper in the rollout. A simple fix is to randomly select an observation in recent history (not just the most recent one) to replace with a sample. Specifically, randomly select an offset $j > 0$ and replace the observation at step $t-j$ with a sample $\hat{\mathbf{o}}_{t-j}^i \sim T_M(\cdot \mid \mathbf{c}_{t-j}^i)$ to generate the hallucinated context $\hat{\mathbf{c}}_t^i = \mathbf{c}(h_{t-j-1}^i a_{t-j}^i \hat{\mathbf{o}}_{t-j}^i a_{t-j+1}^i \mathbf{o}_{t-j+1}^i \ldots a_{t-1}^i \mathbf{o}_{t-1}^i a_t^i)$. If the model is Markovian (that is, if its context only considers the most recent observation), then these two strategies are equivalent. In the experiments, the model is 2nd-order Markov, so j is either 1 or 2.

Deep Context: The above approaches have the drawback that they only introduce a single sampled observation into the context, whereas the contexts encountered by the model during a rollout will largely consist *entirely* of sampled observations. Even if the context only makes use of one observation, the above approaches sample that observation as if it is the first in the rollout, whereas the contexts encountered by the model will largely be sampled after a long rollout. To address this, randomly select a rollout length l and sample observations $\hat{\mathbf{o}}_j^i$ for $j = t-l \ldots t-1$. Then the hallucinated context is

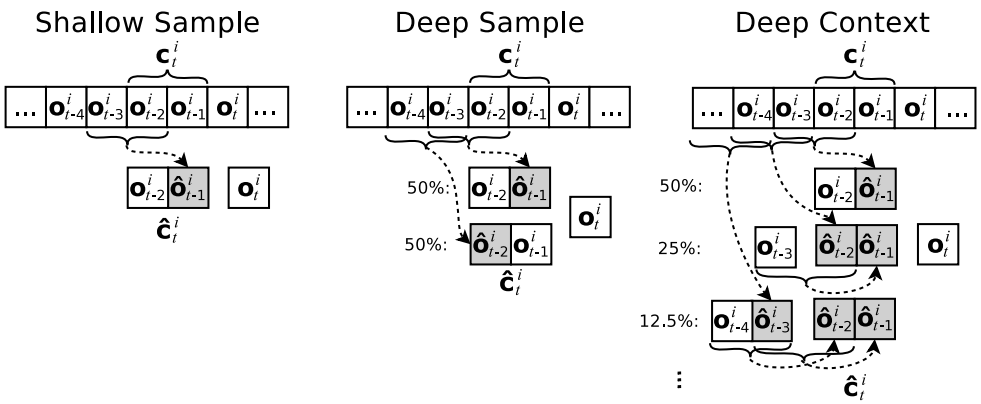

Figure 3: Methods for producing hallucinated contexts, illustrated with a 2nd-order Markov context. Samples are shaded; dashed arrows indicate the contexts which generated the samples. Actions are not shown.

$\hat{\mathbf{c}}_t^i = \mathbf{c}(h_{t-l-1}^i a_{t-l}^i \hat{\mathbf{o}}_{t-l}^i \dots a_{t-1}^i \hat{\mathbf{o}}_{t-1}^i a_t^i)$. In the experiments below, $l > 0$ is drawn with probability $\Pr(l) = \frac{1}{2^l}$. This biases the distribution toward short rollouts, since long rollouts are expected to produce very noisy observations.

3.1 IMPORTANCE OF DETERMINISM

Note that hallucinated replay as described can not sensibly be applied to stochastic environments. In a deterministic environment, the only reason $\hat{\mathbf{c}}_t^i$ can differ from $\mathbf{c}_t^i$ is that the model is imperfect. In a stochastic environment, even a perfect model could generate $\hat{\mathbf{c}}_t^i \neq \mathbf{c}_t^i$ by sampling a different stochastic outcome. In that case, training on the pair $\langle \hat{\mathbf{c}}_t^i, \mathbf{o}_t^i \rangle$ could harm the model by contradicting temporal dependencies the model may have correctly identified. One exception to this limitation is when the source of stochasticity is "measurement error," noise that is uncorrolated over time and does not affect the evolution of the underlying state. In that case, hallucinated replay would re-sample the noise to no ill effect. This work focuses on the case of stochastic models of deterministic environments (assuming the environment may be too complex to model perfectly). This is, in itself, a large and interesting class of problems. The possibility of extending the idea to stochastic environments is briefly discussed in Section 5.2.

4 EXPERIMENTS

This section contains the main results of the paper, an empirical exploration of the properties of hallucinated replay under different modeling conditions. The experiments will be performed on variations of the Mini-Golf problem described in the introduction. One of the key features of this domain is that, though it is a simple planning problem (the agent needs only to select *hit* at the right moment), a long rollout is necessary to sample the eventual reward value and determine whether it is better to *hit* or *no-op*. The experiments employ the simple *1-ply Monte Carlo* algorithm: at each step, for each available action a, perform 50 rollouts of length at most 80 that begin with a and uniformly randomly select actions thereafter. After the rollouts, execute the action that received the highest average return. In these experiments model learning and planning are both performed online – at each step, the planner uses the current model, which is then updated with the resulting training pair. All of the replay strategies train on 5 additional pairs from past experience per step.

The model of the pixels is factored, as described in the introduction. If a neighborhood contains pixels outside the boundaries of the image they are encoded with a special "out of bounds" color. In all the experiments data is shared across all positions, providing some degree of positional invariance in the predictions. To predict the reward and termination signals r_t and ω_t (which do not have positions on the screen) the context is the pixel values at all positions in $\mathbf{o}_t$ and $\mathbf{o}_{t-1}$, as well as the actions a_t and a_{t-1}. During sampling the pixel values are sampled before reward and termination so the sampled image can be supplied as input.

In most of the experiments below the *Context Tree Weighting* (CTW) algorithm (Willems et al. 1995) is used to learn the component models, similar to the FAC-CTW algorithm (Veness et al. 2011), which was also applied in a model-based reinforcement learning setting. Very briefly, CTW maintains a Bayesian posterior over all prunings of a fixed decision tree. Amongst its attractive properties are computationally efficient updates, a lack of free parameters, and guaranteed zero asymptotic regret with respect to the maximum likelihood tree in the model class. In order to apply CTW, an ordering must be selected for the variables in the context tree. In all cases the actions and observations were ordered by recency. For the pixel models, neighboring pixels within a timestep were ordered by proximity to the pixel being predicted. For the reward and termination models, pixel values were in reverse column-major order (bottom-to-top, right-to-left).

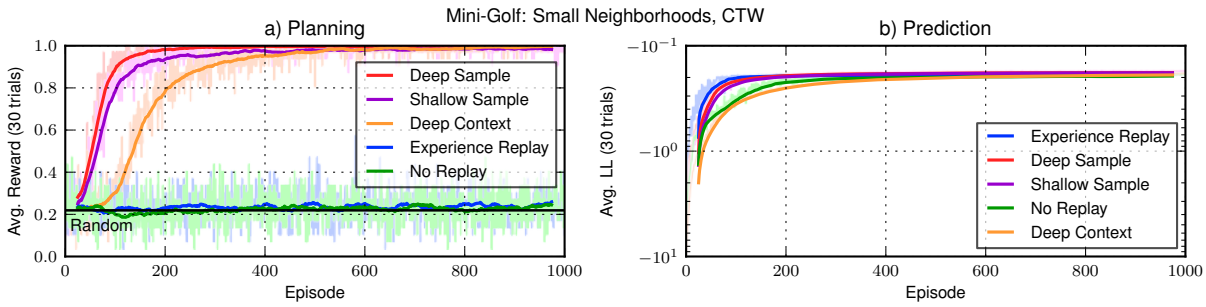

Figure 4: CTW results in Mini-Golf with small neighborhoods, averaged over 30 independent trials. Curves are smoothed using a sliding average of width 50. Unsmoothed data is shown in faint colors. The legends list the replay method in decreasing order of the value of the curve at episode 200 (in both cases, higher is better). Note the log scale in the prediction performance graph. These conventions are maintained in subsequent figures.

4.1 CORRECTING SAMPLE ERRORS

The first experiment considers the setting described in the introduction in which models are learned with 7×3 neighborhoods. Figure 4 shows the results of training with the three hallucinated replay variations proposed above and training with pure "Experience replay" (re-training on randomly selected training pairs from past experience without modifying the context). The results of training with only the agent's experience ("No Replay") are also provided as a baseline to show the effects of replay alone.

The most dramatic result can be seen in the planning performance (Figure 4a): the models trained only with experience from the environment (regardless of whether replay was used) result in behavior that is no better than an agent that chooses actions uniformly randomly (indicated by the solid black line). As can be seen in Figure 4b, the models trained using hallucinated replay achieve no better (and in one case much worse) prediction accuracy, as measured by average per-step log-likelihood over the training data (not counting replay). Nevertheless, they all result in near perfect planning performance. This matches the intuition described above, and indeed the rollouts in Figure 2 were generated using these learned models.

It is unsurprising that the Deep Sample replay strategy would outperform the Shallow Sample strategy – it was indeed important that sampled observations appear in both positions in the context. It is somewhat surprising that Deep Sample outperformed Deep Context as well. The hypothesis that it would be important for the model to be trained on contexts sampled far into a rollout is not supported by these results. It may be that the noisy samples obtained from long rollouts, especially early in training, were misleading enough to harm performance. It is also possible that this effect is an artifact of this example. Nevertheless, these qualitative findings were consistent across the variations explored below so, to reduce clutter, only Deep Sample results are reported in the subsequent experiments.

4.2 FULL NEIGHBORHOODS

The previous experiment is an example of hallucinated replay making meaningful planning possible when a perfect model is not contained within the model class. This is the most common case in problems of genuine interest. Nevertheless, it is relevant to ask what effect it has when a perfect model *is* in the class.

Figures 5a and 5b show the results in the Mini-Golf problem when the models used 9×3 neighborhoods (sufficient to make accurate predictions). Both the Experience Replay and No Replay models are able to support planning, and experience replay clearly improves sample efficiency. The Hallucinated Replay model has consistently lower prediction accuracy and lags behind the Experience Replay model in planning performance, though it eventually catches up. Noting that the hallucinated contexts in the very beginning of training are likely to be essentially meaningless, Figure 5 also shows the results of performing experience replay for the first 50 episodes, and hallucinated replay thereafter. This model's planning performance is nearly identical to that of the Experience Replay model. Once the model has had some time to learn, the sampled observations become more and more like real observations, so this similarity is not surprising.

Note that though a perfect model is in the class, the intermediate models during learning are imperfect and their sampling errors do impact planning performance. It is interesting that hallucinated replay does not aid planning performance by making the model more robust to these errors. One hypothesis might be that if the rollouts were much longer, errors would be more likely to affect planning, and hallucinated replay might have more of an impact.

Figures 5c and 5d show the results of a variation on Mini-Golf that is the same in every respect, except that there are 6 walls to knock down rather than 1. Thus rollouts must be *much* longer for successful planning. First note that the

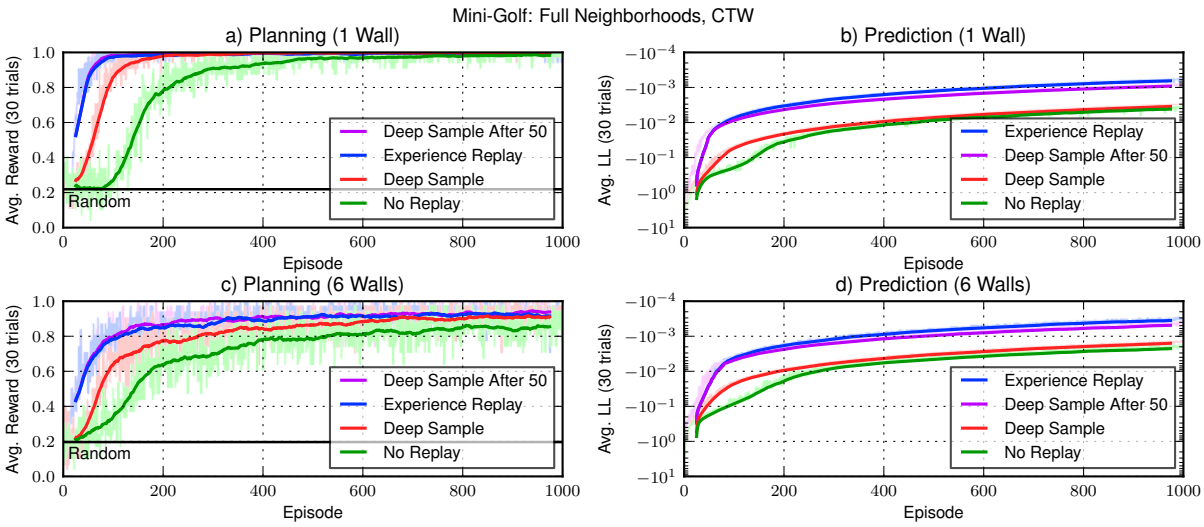

Figure 5: CTW results in Mini-Golf with full neighborhoods, both with 1 wall and with 6 walls.

hypothesis that longer rollouts will cause more sampling errors is borne out. For a comparable level of prediction accuracy, the models achieve substantially better planning performance in the 1 wall version than the 6 wall version. Still, hallucinated replay was not able to mitigate the impact of the model errors.

The errors produced by these "models-in-training" tend to be "salt-and-pepper" noise caused by pixel models that all assign small but positive probability to the incorrect outcome. This is a relatively unstructured type of noise in comparison to that seen in the first experiment. It may simply be that more data is required to learn to correct for the noise than to simply improve accuracy. Or, since objects in this problem consist of only a few pixels, this type of noise may be too destructive to be compensated for.

4.3 A DIFFERENT MODEL CLASS

In order to validate that the benefits of hallucinated replay are robust to the choice of model class, Figure 6 shows the results of training feed forward neural networks in the Mini-Golf domain. The same model architecture was used except here the pixel, reward, and termination models are neural networks with 10 logistic hidden units and a logistic output layer, trained using standard backpropagation (Rumelhart et al. 1986). Several values of the learning rate α were tested and the results using the best value of α for each method are presented. Figures 6a and 6b show the planning and prediction performance of the neural network models when given 7×3 neighborhoods. The results are very similar to those using CTW models. Figures 6c and 6d show the results using 9×3 neighborhoods. Here, unlike with CTW, the hallucinated replay training does not seem to cause a significant loss in performance. The neural network is likely less sensitive to the initially uninformative hallucinated samples because it is easily able to ignore irrelevant features.

Notably, waiting to start hallucinated replay *hurts* performance rather than helps. The reason for this can be seen in the prediction performance graphs: a large drop in prediction performance is observed when hallucinated replay begins – this is because the hidden units must adjust to a sudden change in the distribution of input vectors. This indicates that the compatibility of a model learning method with hallucinated replay may depend on its ability to gracefully handle the non-stationarity it creates in the training data. No-regret algorithms like CTW may be particularly well-suited for that reason.

Neural network models also provide an opportunity to examine another source of model error. Figure 7 shows the results (planning only) of using neural network models with too few hidden nodes (but with full neighborhoods). In Figure 7a, the networks have 5 hidden nodes. The models trained with experience replay are able to learn quickly, but level off at sub-optimal behavior. Training with hallucinated replay is slower, but planning performance ultimately surpasses that of the experience replay model. In Figure 7b, the average score of the last 100 episodes (out of 5000) is shown for various numbers of hidden nodes. The best learning rate for each method and each number of hidden nodes is used. With a small number of hidden nodes, the model is unable to make good predictions or correct for errors. With a large number, an accurate model can be learned and hallucinated replay is unnecessary. For intermediate values hallucinated replay is able to compensate somewhat for the model's representational deficiency.

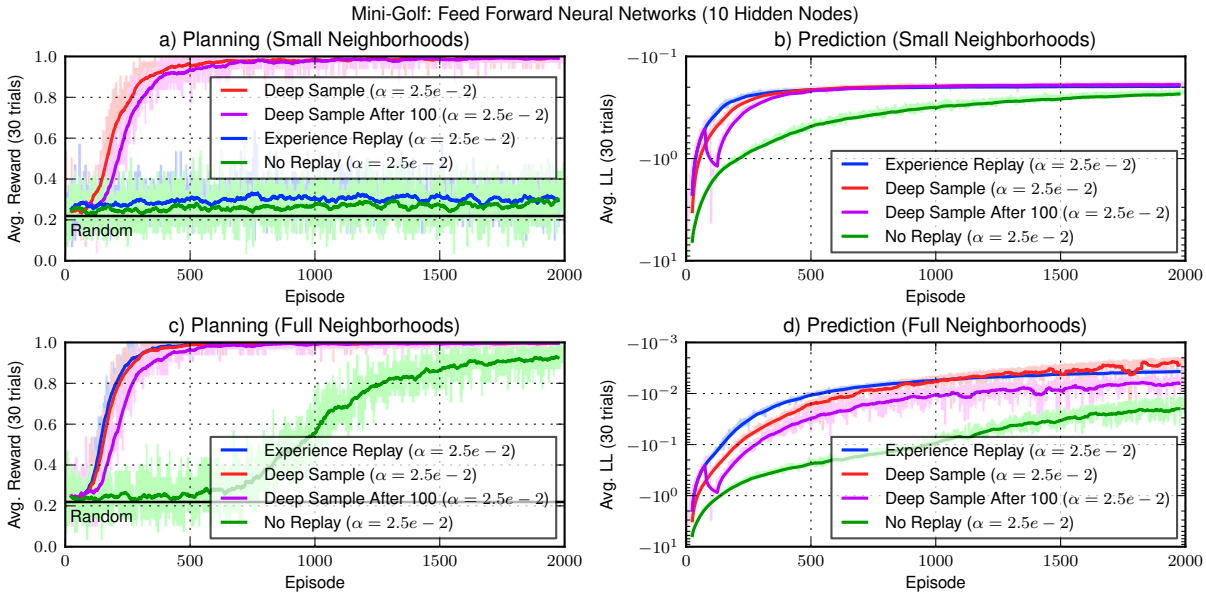

Figure 6: Neural network results in Mini-Golf with 10 hidden nodes.

4.4 ADDITIONAL EXAMPLES

Finally, as additional examples of model impairments that hallucinated replay can help to overcome, consider two variations on the Mini-Golf domain.

4.4.1 DECEPTIVELY INFORMATIVE FEATURES

In the first variation, a score display is added. The observations are now 20×6 images. The bottom three rows of the image are blank for the most part, but at the end of an episode the 3×3 area in the bottom right displays a digit ("0" or a "1") to show the reward received.

As can be seen in Figure 8a, when the model is trained using only real experience this seemingly innocuous addition causes severe planning problems, even though the pixel models are given sufficient contexts to perfectly model the dynamics of the game. The problem arises because the neighborhood size is *not* sufficient to model the score display. When trained only on real experience, the reward and termination models rightly learn that the score display is a reliable predictor. However, during rollouts, those pixels no longer behave as they do in the world, and the reward and termination models make erratic predictions.

In contrast, the models trained with hallucinated replay learn that the score display pixels cannot be relied upon and focus instead on other relevant contextual information (such as the ball's position). This allows the models to provide meaningful predictions that result in good planning performance. Note again that hallucinated replay has not fixed the errors. The hallucinated replay models are no better at predicting what the score display will do. They are able to make good predictions about the reward and termination signals *despite* sample errors in the score display.

4.4.2 COMPLICATED, IRRELEVANT FEATURES

In the second variation, an irrelevant but hard-to-predict component is added to the image. Specifically, the images are 21×3 where the last column displays a set of "blinking lights." Only one pixel in the last column is white on any given step, but the pattern of which pixel is white is 3rd-order Markov. Since the model is 2nd-order Markov, it cannot reliably predict the light pattern. That said, the lights have no impact on the rest of the dynamics and the models have sufficient context to make perfect predictions for every other pixel. Nevertheless, as can be seen in Figure 8b, this addition once again causes the model trained only on real experience to fail when used for planning.

In this case the problem is that rollout samples of the blinking lights will not resemble those seen in the world. In the world, only one light can be on a time but in samples lights stochastically turn on and off, causing novel configurations. The pixel models neighboring these unfamiliar configurations have higher uncertainty as a result, and the now familiar error compounding process ensues, corrupting the important parts of the sampled image as well as the unimportant parts. The models trained using hallucinated replay, in contrast, learn to make predictions given the distribution of light configurations that will actually be encountered during a rollout.

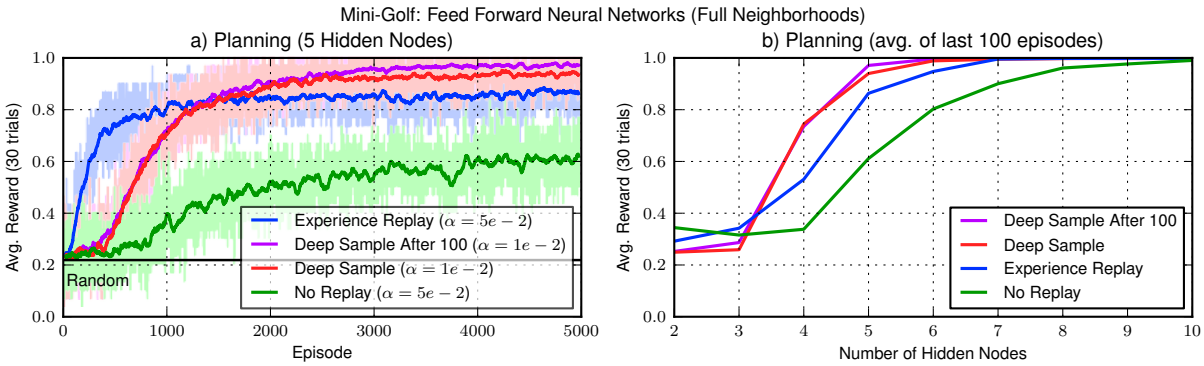

Figure 7: Neural network results in Mini-Golf with full neighborhoods but too few hidden nodes.

5 DISCUSSION

The results have shown that even mildly limited factored models that can accurately predict nearly every aspect of the environment's dynamics can dramatically fail when used in combination with Monte-Carlo planning. Since factored models and Monte-Carlo planning are both important, successful tools in problems with large state/observation spaces, resolving this issue may be a key step toward scaling up model-based reinforcement learning. Overall, hallucinated replay was effective in its purpose: to train models to be more robust to their own sampling errors during rollouts.

When a perfect model was contained within the class (a rare occurrence in problems of genuine interest), hallucinated replay was comparable to (but slightly less effective than) pure experience replay. Though in this case the models are imperfect during training, hallucinated replay did not seem to be effective at compensating for the type of wide-spread, unstructured noise that resulted from transient parameter settings (as opposed to the more systematic errors resulting from class limitations in other examples).

It was also observed that when the model class is *extremely* limited (e.g. the neural network models with only 2 hidden nodes), hallucinated replay may be ineffective at compensating for errors because the model simply cannot learn to make meaningful predictions at all. Of course, in this case the model is ineffective no matter how it is trained. It seems that hallucinated replay is most effective when the model can capture *some but not all* of the environment's dynamics. When a model is performing poorly hallucinated replay may be able to magnify the impact of improving the model's expressive power (as was seen when the neural networks were given more hidden nodes).

5.1 RELATED WORK

Other authors have noted that the most accurate model in a limited class may not be the best model for planning. Sorg, Singh & Lewis (2010) argue that when the model or planner are limited, reward functions other than the true reward may lead to better planning performance. Sorg, Lewis & Singh (2010) use policy gradient to learn a reward function for use with online planning algorithms. Joseph et al. (2013) make a similar observation regarding the dynamics model and use policy gradient to learn model parameters that lead to good planning, rather than low prediction error (demonstrating that the two do not always coincide). The results presented here add to the evidence that simply training to maximize one-step prediction accuracy may not yield the best planning results.

Siddiqi et al. (2007) addressed the problem of learning stable dynamics models of linear dynamical systems. The key idea was to project the learned dynamics matrix into the constrained space of matrices with spectral radius of at most 1, and which therefore remain bounded even when multiplied with themselves arbitrarily many times. Though certainly conceptually related (their work has the same goal of learning a model whose predictions are sensible when given its own output as input), their specific approach is unlikely to apply to the setting considered here, as the concept of "stability" is not so easily quantified.

There is a long history in the supervised learning setting of adding noise to training data (or otherwise intentionally corrupting it) to obtain regularization and generalization benefits (see e.g. Matsuoka 1992, Burges & Schölkopf 1997, Maaten et al. 2013). There the issue is not typically prediction composition but more broadly the existence of inputs that the training set may not adequately cover. Without access to the true distribution over inputs, noise is typically generated via some computationally or analytically convenient distribution (e.g. Gaussian) or via a distribution that incorporates prior knowledge about the problem. In contrast, we have access to the model which generates its own inputs, and can thus sample corrupting noise from a learned, but more directly relevant distribution.

Ross & Bagnell (2012) observed a similar train/test mis-

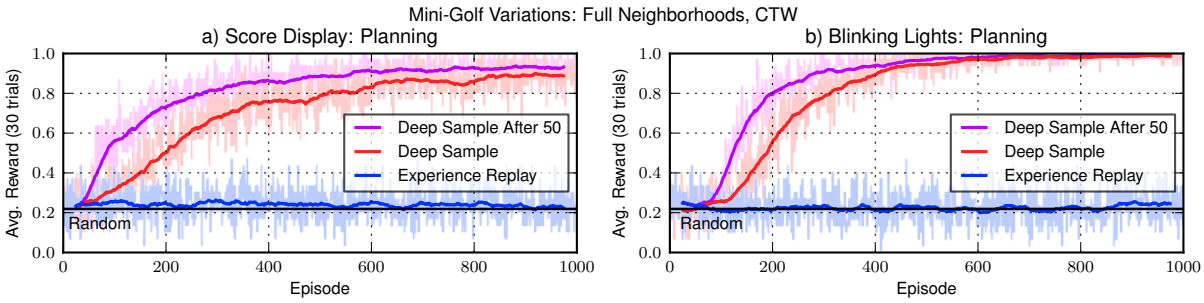

Figure 8: CTW results in two variations of Mini-Golf (described in text).

match in the model-based reinforcement learning setting when the model is trained on a fixed batch of data. In that case, the plan generated using the model may visit states that were underrepresented in training, resulting in poor performance. Their algorithm DAgger mixes experience obtained using model-generated plans into the training data as the model learns, thereby closing the gap between training and test distributions. Roughly speaking, they prove that their approach achieves good planning performance when a model in the class has low prediction error. In the examples considered here *no* model in the class has low prediction error, but near perfect planning performance is nevertheless possible.

5.2 FUTURE DIRECTIONS

The results indicate that, in addition to one-step accuracy, a model's robustness to its own errors should be a key concern for model based reinforcement learning. They also indicate that training a model on its own outputs is a promising approach to addressing this concern. These observations open the door to a number of interesting questions and issues for further study.

As discussed in Section 3, hallucinated replay is not immediately applicable in stochastic domains. While a lot of perceived stochasticity can be attributed to unmodeled (but deterministic) complexity, it would nevertheless be valuable to explore whether this approach can be applied to inherently stochastic dynamics. The main challenge in this case is forcing the model's hallucinated data to choose the same stochastic outcome that was observed in the real data. It may be that this could be accomplished by learning reverse correlations from real world outcomes to hallucinated contexts and applying rejection sampling when generating hallucinated data.

The hallucinated replay algorithm presented in this paper is simple and offers no theoretical guarantees. It may, however, be possible to incorporate the basic idea of hallucinated replay into more principled algorithms that admit more formal analysis. For instance, it may be possible to extend the DAgger algorithm given by Ross & Bagnell (2012) using a form of hallucinated replay. This would allow the analysis to take into account the learned model's robustness (or lack thereof) to its own errors. It would also be valuable, if possible, to develop algorithms that have the benefits of hallucinated replay but are able to promise that the hallucinated data will not harm asymptotic performance if an accurate model is in the class.

Perhaps most important would be to develop a more formal understanding of the relationships between accuracy, robustness, and planning performance. In sufficiently interesting environments, model errors will be inevitable. Successfully planning despite those errors is key to applying model based reinforcement learning to larger, more complex problems. A characterization of the properties beyond simple accuracy that make a model good for planning and a more nuanced understanding of which types of model error are acceptable and which are catastrophic could yield improved forms of hallucinated replay, or other model learning approaches that have good planning performance as an explicit goal rather than an implicit effect of accuracy.

6 CONCLUSIONS

Hallucinated replay trains a model using inputs drawn from its own predictions, making it more robust when its predictions are composed. This training methodology was empirically shown to substantially improve planning performance compared to using only real experience. The results suggest that hallucinated replay is most effective when combined with model-learning methods that gracefully handle non-stationarity and when the model class' limitations allow it to capture some but not all of the environment's dynamics.

Acknowledgements

Thank you to Michael Bowling, Marc Bellemare, and Joel Veness for discussions that helped shape this work. Thanks also to Joel Veness for his freely available FAC-CTW implementation (`http://jveness.info/software/mc-aixi-src-1.0.zip`).

References

Burges, C. J. & Schölkopf, B. (1997), Improving the accuracy and speed of support vector machines, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 375–381.

Joseph, J., Geramifard, A., Roberts, J. W., How, J. P. & Roy, N. (2013), Reinforcement learning with misspecified model classes, *in* '2013 IEEE International Conference on Robotics and Automation (ICRA)', pp. 939–946.

Kearns, M., Mansour, Y. & Ng, A. Y. (2002), 'A sparse sampling algorithm for near-optimal planning in large markov decision processes', *Machine Learning* **49**(2-3), 193–208.

Kocsis, L. & Szepesvári, C. (2006), Bandit based monte-carlo planning, *in* 'Proceedings of the 17th European Conference on Machine Learning (ECML)', pp. 282–293.

Lin, L.-J. (1992), 'Self-improving reactive agents based on reinforcement learning, planning and teaching', *Machine learning* **8**(3-4), 293–321.

Maaten, L., Chen, M., Tyree, S. & Weinberger, K. Q. (2013), Learning with marginalized corrupted features, *in* 'Proceedings of the 30th International Conference on Machine Learning (ICML-13)', pp. 410–418.

Matsuoka, K. (1992), 'Noise injection into inputs in back-propagation learning', *IEEE Transactions on Systems, Man and Cybernetics* **22**(3), 436–440.

Ross, S. & Bagnell, D. (2012), Agnostic system identification for model-based reinforcement learning, *in* 'Proceedings of the 29th International Conference on Machine Learning (ICML-12)', pp. 1703–1710.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), 'Learning representations by back-propagating errors', *Nature* **323**(9), 533–536.

Siddiqi, S. M., Boots, B. & Gordon, G. J. (2007), A constraint generation approach to learning stable linear dynamical systems, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 1329–1336.

Silver, D. & Veness, J. (2010), Monte-carlo planning in large pomdps, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 2164–2172.

Sorg, J., Lewis, R. L. & Singh, S. (2010), Reward design via online gradient ascent, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 2190–2198.

Sorg, J., Singh, S. & Lewis, R. L. (2010), Internal rewards mitigate agent boundedness, *in* 'Proceedings of the 27th International Conference on Machine Learning (ICML-10)', pp. 1007–1014.

Veness, J., Ng, K. S., Hutter, M., Uther, W. T. B. & Silver, D. (2011), 'A Monte-Carlo AIXI Approximation', *Journal of Artificial Intelligence Research* **40**, 95–142.

Willems, F. M., Shtarkov, Y. M. & Tjalkens, T. J. (1995), 'The context tree weighting method: Basic properties', *IEEE Transactions on Information Theory* **41**, 653–664.