Learning Approximate Stochastic Transition Models

Yuhang Song[1], **Christopher Grimm**[2], **Xianming Wang**[3], **Michael L. Littman**[4]
[1]**Beijing University of Aeronautics and Astronautics**, [2]**University of Michigan**,
[3]**Renmin University of China**, [4]**Brown University**
yuhangsong@buaa.edu.cn, crgrimm@umich.edu, xianmingwang@ruc.edu.cn, michael_littman@brown.edu

Abstract

abs

Intoduction

Model-based approaches separate the reinforcement-learning (RL) problem into two components. The first component learns a transition model that predicts the next state from the current state and action. The second component uses that model to make decisions by looking ahead to predict the consequences of different courses of actions. This paper focuses on the first problem of acquiring the model, specifically addressing the development of a mechanism for learning to approximate a stochastic transition function.

Background

A Markov decision process (MDP) model of an environment consists of a set of states S and actions A, a transition function $T : S \times A \to \Pi(S)$ mapping state–action pairs to a probability distribution over next states, and a reward function $R : S \times A \to \Re$.

Since the focus of this paper is not on decision making but on learning the transition function, we simplify the transition function to $T(s, s')$, which represents the probability that state s' will follow s. A separate function can be learned for each action $a \in A$. Although some authors have found there to be an advantage to representing the transitions jointly for all actions (cite Satinder paper), this issue is orthogonal to the representation issue we address here.

To review methods for learning $T(s, s')$, we begin by separating out three representations for transition models. A *query* model is one that can answer, for any s, s' pair, the probability of s' given s. Such a model can be represented as a table for the state space is relatively small (reference). It can also be captured by a dynamic Bayesian network (degris? others?). Some types of planners, such as ones based on policy iteration (cite Puterman, cite older DBN work), require access to these probabilities to compute expected values. Query models can be very challenging to work with and learn when the size of the state space if enormous, however, because looping over all the possible values of s' can be too expensive. It is especially problematic when most s, s' pairs have zero probability, since considering each of them is expensive and pointless.

A *sparse* model is a refinement of the query method that takes a state s as input and returns a list of states $N(s)$ such that $s' \in N(s)$ iff $T(s, s') > 0$. Such a representation can be used much more efficiently as it only needs to consider the non-zero entries of $T(s, s')$. Tabular and DBN methods can be used in this setting, but a general approach has yet to be articulated. In addition, they provide no advantage of the query model in environments in which $|N(s)|$ is intractably large.

An alternative is a *generative* model, which is a function G that, given, s, produces s' Learning a generative model is a more classical supervised learning problem. Given examples of s, s' pairs, learn a mapping such that $G(s)$ produces s'. When the target mapping is deterministic, many learning algorithms can be brought to bear. (Cite Traviste, Singh, other deep learning model-based work, old work from Moore and Atkeson) These learning algorithms can be applied to stochastic transition models, but, as we show, there are significant pitfalls to doing so. An exception is in control problems where the transition model has the form $s' \sim G(s) + \eta$, where the η is state independent and typically small and zero mean noise, so that planning using $G(s)$ results in an approximation to planning with the stochastic model—the noise can be safely ignored during planning (probably some citations—LQR?).

(Need to cite: UCT paper, sparse sampling paper, work on planning using Bayes net representation).

Background

GANs WGANs

Model Stochastic Transition with SGAN

To model stochastic transition in model-base RL with generative models, the basic paradigm is having a *Generator* G, which takes in current state $\bar{x}$ and a random noise $\mathfrak{n}$, and generate a possible next state x_g,

$$x_g = G^{\mu}(\bar{x}, \mathfrak{n}) \tag{1}$$

where μ is the parameter to learned in G. Assuming x_g under $\bar{x}$ follows the distribution $\mathbb{P}_g^{\bar{x}}$, while real transition x_r under $\bar{x}$ follows the distribution $\mathbb{P}_r^{\bar{x}}$, the ultimate goals are tow-folders,

- x_g has higher probability to be a valid state. For example, for 2D grid world, generate less next state with two agents or blurry agent. This goal is measured and reported in this paper as valid percentage VP.
- $\mathbb{P}_g^{\bar{x}}$ should approach $\mathbb{P}_r^{\bar{x}}$ as close as possible. This goal is measured and reported in this paper with L1 loss.

To achieve this, there is another model called *Discriminator* D, which takes in current state $\bar{x}$ and generated next state x_g or real next state x_r, and produces a scalar to evaluate if the input transition pair is 'good' under the concept of a specific distance,

$$distance = D^{\theta}(\bar{x}, \{x_r \text{ or } x_g\}) \tag{2}$$

where μ is the parameter to learned in G. Based on the d-ifferent distance D are measuring, different GAN The first goal is a in-build goal for traditional Generative adversarial network (GAN), in which the state-of-the-art Wasserstein GAN with gradient penalty (GP-WGAN) can achieve the best VP. However, for the traditional GANs, the second goal is not specially addressed theoretically, as well as verified to be poor in out experiments. This motivates us to propose a novel distance

The Proposed S Distance

ML: I think, we should define the algorithm first, then introduce these concepts to help explain it. Explaining it first, without saying what we're working up toward, is confusing.

We consider the one dimensional case as a start, where x_r are real samples sampled from distribution $\mathbb{P}_r$, and x_g are generated samples sampled from distribution $\mathbb{P}_g$,

$$x_r \sim \mathbb{P}_r \tag{3}$$

$$x_g \sim \mathbb{P}_g \tag{4}$$

Note that both x_r and x_g are restricted between $[0, 1]$. Following is the proposed S distance,

$$S(\mathbb{P}_r, \mathbb{P}_g) = \mathbb{E}_{x_g \sim \mathbb{P}_g}\{|\int_{x_g}^{1} \mathbb{P}_r(x)dx - \int_{x_g}^{1} \mathbb{P}_g(x)dx|\} \tag{5}$$

While, the Wasserstein distance is defined to be,

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \{\mathbb{E}_{x_r \sim \mathbb{P}_r}[f(x_r)] - \mathbb{E}_{x_g \sim \mathbb{P}_g}[f(x_g)]\} \tag{6}$$

where f belongs to a set of 1-Lipschitz functions. Apparently, both S and W distance will be minimized if the $\mathbb{P}_r$ and $\mathbb{P}_g$ are identical. In a GAN paradigm, *Generator* G updates itself at each sample x_g to minimize the distance they are based on. To take a deeper insight of the advantage of the proposed S distance, we consider what G is trying to minimize at each sample x_g, when based on S and W distance respectively. When using S distance, G at x_g is minimizing,

$$S_{\mathbb{P}_r, \mathbb{P}_g}(x_g) = |\int_{x_g}^{1} \mathbb{P}_r(x)dx - \int_{x_g}^{1} \mathbb{P}_g(x)dx| \tag{7}$$

Algorithm 1 SGAN. We use default values of $C = 5$, $B = 32$, $\alpha = 0.0001$, $\beta_1 = 0.0$, $\beta_2 = 0.9$. And δ is computed for different domains.

Require: The number of critic iterations per generator iteration C, the batch size B, Adam hyper-parameters α, β_1, β_2, hyper-parameter δ.
Require: Initial *Discriminator* D parameters θ_0, initial *Generator* G parameters μ_0.

Load offline-DHP model to initialize model with parameter vectors: $\{\theta_{\hat{\nu}}, \theta_{\hat{\pi}}, \theta_V\}$;
while θ has not converged **do**
  for $c = 1, \cdots, C$ **do**
    for $b = 1, \cdots, B$ **do**
      Sample real transition pair $\bar{x} \rightarrow x_r$, where $x_r \sim \mathbb{P}_r^{\bar{x}}$.
      Sample latent variable $\mathfrak{n} \sim U[0, 1]$.
      $x_g \leftarrow G^{\mu}(\bar{x}, \mathfrak{n})$
      $d \leftarrow \|x_r - x_g\|$
      $T_b \leftarrow d/\delta$
      for $t = 1, \cdots, T_b$ **do**
        Sample $\tau \sim U[0, 1]$
        $x_\tau \leftarrow \tau x_r + (1 - \tau)x_g$
        $L_d^{(b,t)} \leftarrow \|\nabla_{x_\tau} D^{\theta}(\bar{x}, x_\tau) - \frac{x_r - x_g}{\|x_r - x_g\|}\|^2\}$
      end for
    end for
    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \frac{1}{T_b} \sum_{t=1}^{T_b} L_d^{(b,t)}, \theta, \alpha, \beta_1, \beta_2)$
  end for
  for $b = 1, \cdots, B$ **do**
    Sample latent variable $\mathfrak{n} \sim U[0, 1]$.
    $L_g^{(b)} \leftarrow -D^{\theta}(G^{\mu}(\bar{x}, \mathfrak{n}))$
  end for
  $\mu \leftarrow \text{Adam}(\nabla_\theta \frac{1}{B} \sum_{b=1}^{B} L_g^{(b)}, \mu, \alpha, \beta_1, \beta_2)$
end while

while using W distance, G at x_g is minimizing,

$$W_{\mathbb{P}_r,\mathbb{P}_g}(x_g) = f(x_g) \approx \mathbb{P}_r(x_g) - \mathbb{P}_g(x_g) \tag{8}$$

We can see that $S_{\mathbb{P}_r,\mathbb{P}_g}(x_g)$ consider how unbalance are the two distributions in a whole sight, while the $W_{\mathbb{P}_r,\mathbb{P}_g}(x_g)$ considers the unbalance of the two probabilities at this specific x_g. One can easily think of a x_g, where $W_{\mathbb{P}_r,\mathbb{P}_g}(x_g)$ is zero, but the distributions $\mathbb{P}_r$ and $\mathbb{P}_r$ are not identical, which means $W_{\mathbb{P}_r,\mathbb{P}_g}(x_g)$ is failing. But under this situation, $S_{\mathbb{P}_r,\mathbb{P}_g}(x_g)$ still gives a right direction for x_g to update, since it observes the unbalance of $\mathbb{P}_r$ and $\mathbb{P}_r$ on each side of the x_g.

GAN Based on S Distance

Following we proposed the method to achieve this S distance in a GAN. We still describe things in one-dimensional case to make it simple and straight forward. For every x_r, x_g pair, we sample x_τ between x_r and x_g,

$$x_\tau = \tau x_r + (1-\tau)x_g \tag{9}$$

where

$$\tau \sim U[0,1] \tag{10}$$

Following, we consider our problem on a discrete space with interval of $\varepsilon \to 0$, we give every notation of x a check mark, i.e., $\check{x}$, to mark that they are discrete value under interval ε. Later on, we will derive limitation on $\varepsilon \to 0$, so that we can have a general conclusion on the continuous space. Now consider a event denoted by: $\check{x}_\tau \overset{T}{=} \check{x}_n$, which means,

- Sample $\check{x}_\tau$ for T times, $\check{x}_n$ got sampled as $\check{x}_\tau$ at least for one time.

To be clear, $\check{x}_\tau$ $\check{x}_r$, $\check{x}_g$ are all random variables, and $\check{x}_n$ is a specific point. Apparently, we have,

$$P(\check{x}_\tau \overset{1}{=} \check{x}_n|\check{x}_r,\check{x}_g) = \begin{cases} \frac{1}{d/\varepsilon} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{else} \end{cases} \tag{11}$$

where

$$d = |\check{x}_r - \check{x}_g| \tag{12}$$

If we sample $\check{x}_\tau$ for T times, where

$$T = d/\delta \tag{13}$$

Then, we have,

$$\begin{aligned} & P(\check{x}_\tau \overset{T}{=} \check{x}_n|\check{x}_r,\check{x}_g) \\ = \; & 1-(1-P(\check{x}_\tau \overset{1}{=} \check{x}_n|\check{x}_r,\check{x}_g))^t \\ = \; & \begin{cases} 1-(1-\frac{1}{d/\varepsilon})^{d/\delta} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{else} \end{cases} \end{aligned} \tag{14}$$

Apparently, δ has to satisfy,

$$\delta = z\varepsilon \tag{15}$$

where $z \in Z^+$, since ε is the minimal value a computer can operate, and δ has to be a positive integer multiple of ε. And this z is a fixed value as a hyper-parameter. Now, we consider following limit,

$$\begin{aligned} & \lim_{\delta=z\varepsilon,\varepsilon\to 0}(1-\frac{1}{d/\varepsilon})^{d/\delta} \\ = \; & \lim_{\delta=z\varepsilon,\varepsilon\to 0} e^{d/\delta \ln(1-\frac{1}{d/\varepsilon})} \\ = \; & \lim_{\delta=z\varepsilon,\varepsilon\to 0} e^{\frac{\ln(\frac{d-\varepsilon}{d})}{\delta/d}} \\ = \; & \lim_{\varepsilon\to 0} e^{\frac{\frac{d}{d-\varepsilon}\frac{-1}{d}}{z/d}} \\ = \; & e^{-1/z} \end{aligned} \tag{16}$$

Put the conclusion of (16) into (14), we have,

$$\begin{aligned} & P(x_\tau \overset{T}{=} x_n|x_r,x_g) = \lim_{\varepsilon,\delta\to 0} P(\check{x}_\tau \overset{T}{=} \check{x}_n|\check{x}_r,\check{x}_g) \\ = \; & \begin{cases} 1-e^{-1/z} & x_r < x_n < x_g, x_g < x_n < x_r \\ 0 & \text{else} \end{cases} \end{aligned} \tag{17}$$

where we have switch back to the continuous space and have this general conclusion. Now, we propose our update rules for the *Discriminator* D with parameter θ to be optimized,

$$\theta \longrightarrow \theta + \nabla_\theta\{-|\nabla_{x_\tau} D^\theta(x_\tau) - \frac{x_r - x_g}{|x_r - x_g|}|^2\} \tag{18}$$

which means we try to make $\nabla_{x_\tau} D^\theta(x_\tau)$ approach $\frac{x_r - x_g}{|x_r - x_g|}$. Lets take a look at $\nabla_{x_\tau} D^\theta(x_\tau)$ at a specific point x_n,

$$\begin{aligned} & \nabla_{x_\tau = x_n} D^\theta(x_\tau = x_n) \\ = \; & \mathbb{E}_{\tau\sim U[0,1], x_r\sim\mathbb{P}_r, x_g\sim\mathbb{P}_g}\{\frac{x_r - x_g}{|x_r - x_g|}\} \\ = \; & P(x_\tau \overset{T}{=} x_n|x_g < x_n < x_r)P(x_g < x_n < x_r) \\ & -P(x_\tau \overset{T}{=} x_n|x_r < x_n < x_g)P(x_r < x_n < x_g) \end{aligned} \tag{19}$$

which means the value of $\nabla_{x_\tau = x_n} D^\theta(x_\tau = x_n)$ is determined by the probability of it gets positive update and negative update. Since (17), we know that

$$P(x_\tau \overset{T}{=} x_n|x_g < x_n < x_r) = 1 - e^{-1/z} \tag{20}$$

$$P(x_\tau \overset{T}{=} x_n|x_r < x_n < x_g) = 1 - e^{-1/z} \tag{21}$$

Put (20) (21) into (19), we have,

$$\begin{aligned} & \nabla_{x_\tau = x_n} D^\theta(x_\tau = x_n) \\ = \; & [P(x_g < x_n < x_r) - P(x_r < x_n < x_g)](1 - e^{-1/z}) \\ = \; & [P(x_g < x_n)P(x_n < x_r) \\ & -P(x_r < x_n)P(x_n < x_g)](1 - e^{-1/z}) \\ = \; & [\int_0^{x_n} \mathbb{P}_g(x)dx \int_{x_n}^1 \mathbb{P}_r(x)dx \\ & - \int_0^{x_n} \mathbb{P}_r(x)dx \int_{x_n}^1 \mathbb{P}_g(x)dx](1 - e^{-1/z}) \\ = \; & [\int_{x_n}^1 \mathbb{P}_r(x)dx - \int_{x_n}^1 \mathbb{P}_g(x)dx](1 - e^{-1/z}) \end{aligned} \tag{22}$$

Now, we can give the update rule of *Generator* G with parameter μ to be learnt, and we put (22) in this update rule to have a clearer view on what G is doing,

$$\begin{aligned} & \mu \\ \longrightarrow \quad & \mu + \nabla_\mu \{D^\theta(G^\mu(x_g))\} \\ \longrightarrow \quad & \mu + \nabla_\mu \{\frac{\partial D^\theta(G^\mu(x_g))}{\partial G^\mu(x_g)} \frac{\partial G^\mu(x_g)}{\partial \mu}\} \\ \longrightarrow \quad & \mu + \{[\int_{x_g}^{1} \mathbb{P}_r(x)dx \\ & - \int_{x_g}^{1} \mathbb{P}_g(x)dx](1 - e^{-1/z})(\nabla_\mu G^\mu(x_g))\} \end{aligned} \tag{23}$$

which means wherever x_g is, it is updating itself to make $\int_{x_g}^{1} \mathbb{P}_g(x)dx$ approaching $\int_{x_g}^{1} \mathbb{P}_r(x)dx$. One can just take a few cases to confirm this. The absolute error when updating G is actually modelling the proposed S distance in (5) (7).

Between SGAN and Gradient Penalty in WGAN

Gradient penalty WGAN (GP-WGAN) (Gulrajani et al. 2017) has been proposed as the state-of-the-art GAN. In GP-WGAN, they proposed adding a gradient penalty loss to the original loss of D in WGAN, to impose 1-Lipschitz constrain. The gradient penalty is defined as,

$$\theta \longrightarrow \theta + \nabla_\theta \{-(|\nabla_{x_\iota} D^\theta(x_\iota)| - 1)^2\} \tag{24}$$

where x_ι is defined similar as x_τ in our SGAN,

$$x_\iota = \iota x_r + (1 - \iota) x_g \tag{25}$$

$$\iota \sim U[0, 1] \tag{26}$$

The only difference is that they sample x_ι only once for each x_r, x_g pair, but we sample x_τ for T times for each x_r, x_g pair, recall (13). The motivation when they propose this gradient penalty is the optimal D in WGAN,

$$\mathbb{E}_{x_r \sim \mathbb{P}_r, x_g \sim \mathbb{P}_g} \{\nabla_{x_\iota} D^*(x_\iota) = \frac{x_r - x_g}{|x_r - x_g|}\} \tag{27}$$

which is strictly proved in (Gulrajani et al. 2017).

One can see that (27) gives the information on the direction as well as the magnitude of D^* in WGAN. While the gradient penalty in (24) only make a loss of the magnitude regarding to D^*, which raise our question of why not impose both magnitude and direction gradient penalty to the D, so that the D can better approach this D^*. Here we give the gradient penalty that is imposing both direction and magnitude towards D^*,

$$\theta \longrightarrow \theta + \nabla_\theta \{-|\nabla_{x_\iota} D^\theta(x_\iota) - \frac{x_r - x_g}{|x_r - x_g|}|^2\} \tag{28}$$

One can also recall that in our SGAN, the update rule of D in (18) seems to be exactly same as (28), which raise the question of if the proposed SGAN is just doing this optimal D^* in the concept of WGAN.

Both question can be answered when we consider following argument. In the concept of WGAN, the direction of D^* in (27) is actually not achievable. Equation (27) is expressed in a form of expectation, which means if we use gradient penalty as (28), we are updating $\nabla_{x_\iota} D^\theta(x_\iota)$ in diverse directions for all possible x_r, x_g pairs. This further means the final result we have for $\nabla_{x_\iota} D^\theta(x_\iota)$ would rely on the probability of it got updated towards different directions (exactly what we are doing in the concept of SGAN), instead of reaching D^* in (27). We think that is why the gradient penalty in (24) only consider impose the magnitude gradient penalty towards optimal D^* with (24). We also have experiment shows that when impose (28) as a gradient penalty in WGAN, or say, impose both direction and magnitude towards D^*, WGAN is failing.

Apparently, if we try to impose both direction and magnitude towards D^*, we are naturally fall into the concept of S distance, which has been explicit in this paper. And the sampling T times become crucial to track the expectation of $\nabla_{x_\tau} D^\theta(x_\tau)$, as also has been explicit. When we are doing this, the assumptions for WGAN are not holding any more, so we are not imposing a optimal D under W distance concept any more. That is also why we can omit the basic update rules of WGAN's D.

Experimental Comparisons

Comparison algorithms:

1. Tabular learner.
2. (Can we do a Bayes net learner?)
3. Deterministic deep net.
4. WGAN.
5. WGAN with rejection.
6. WGAN with separately learned rejection.

Comparison environments:

1. Random 1d walk: n bits, Pr(right) = 2/3, Pr(left) = 1/3, wrap around.
2. Random 1d flips: n bits. Next state is generated by picking a random bit position and flipping it.
3. Russell Norvig grid dynamics with no obstacles. (Pr up is .8, right is .1, left is .1, down is 0. Grid is maybe 5x5?).
4. Russell Norvig grid dynamics with obstacles.
5. Marbles!

Results

For a, we expect 1 and perhaps 2 to do well. 3 will fail in spite of the simplicity, because it will turn 0s and 1s into fractional values. Not sure about 4, 5, 6, but we're definitely hoping 6 is robust even for large n. We might want to try $n \in \{5, 10, 20\}$.

For b, if we try $n \in \{5, 10, 20\}$, 1 should fail. There are simply too many reachable states to expect to see them. Not sure about 2.

I'm hoping that c and d will have similar results, in which case we probably just want to report d (since it adds the visual element).

Finally, if e works, that's a great way to end.

For a–d, we can measure how good the model is by picking a starting state, running the learned model for k steps, perhaps rounding to the nearest non-fractional representation, and running the real dynamics for k steps and measuring the KL divergence or L1 distance.

RESULTS TABLE.

Note that the table measures the L1 distance between the generated and real distributions. The measurement is conducted with 2000 samples (filtered to 1000 samples), and averaged under 50 times measurement.

E gives about 100% improvement when the domain is image (1D grid and 2D grid). E gives no improvement when the domain is vector. F gives very slight improvement (about 5%) without E. F gives 40% improvement with E under the domain of image. F gives no improvement with E under the domain of vector.

It makes sense that the autoencoder only helps when dealing with images because the autoencoder is correcting the input using an abstraction. You can think of it like this: If the input space is high-dimensional, and the autoencoder has a bottleneck, it has to map a bunch of similar values to the same internal representation before reconstructing. This results in slightly corrupted generated images getting mapped to the same internal representation as their corresponding uncorrupted generated images. In the bitvector case, there is less room for this abstraction to occur, because the input space is small.

Thinking About Loss Functions

Here's my argument for why it should work. Let's say the generator outputs a bunch of possible answers $x_1, \ldots, x_k$. The real distribution assigns these outcomes probabilities of $p_1, \ldots, p_k$, and the generator assigns them $g_1, \ldots, g_k$.

We randomly choose $s \sim \text{Uniform}(\{-1, 1\})$. If $s = 1$, draw $i \sim p_1, \ldots, p_k$. Otherwise, if $s = -1$, $i \sim g_1, \ldots, g_k$. Now, we send x_i to the discriminator. It outputs $d \in \{-1, 1\}$ and incurs a loss of $|d - s| = 1 - ds$. That is, it's guessing whether the source of x_i was the real data (1) or the generator (-1).

The discriminator is a mapping from x_i to $\{-1, 1\}$. That is, we can capture the discriminator with a vector $d_1, \ldots, d_k$. What is the loss for $d_1, \ldots, d_k$?

$$\begin{aligned}
&\frac{1}{2}\sum_i p_i(1 - d_i) + \frac{1}{2}\sum_i g_i(1 + d_i) \\
&= \frac{1}{2}\sum_i (p_i - p_i d_i + g_i + g_i d_i) \\
&= \frac{1}{2}\sum_i d_i(g_i - p_i) + \frac{1}{2}\sum_i p_i + \frac{1}{2}\sum_i g_i \\
&= \frac{1}{2}\sum_i d_i(g_i - p_i) + 1
\end{aligned}$$

What d vector minimizes this loss? It is $d_i = 1$ if $p_i > g_i$, $d_i = -1$ otherwise. That is, when the discriminator sees x_i, it should guess 1 ("real") if the probability that the real distribution assigns to x_i is larger than that assigned to it by the generator.

If it chooses this d vector, the loss becomes:

$$-\frac{1}{2}\sum_i |g_i - p_i| + 1.$$

What g maximizes this loss? Setting $g_i = p_i$ causes the loss to be 1. Any deviation from this choice will decrease the loss. Basically, the discriminator gets an advantage whenever the two probabilities deviate because it can be right more often by choosing the answer that assigns that example higher probability.

Future Work

Planning using the learned models.

References

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.