Prediction Under Uncertainty for Autonomous Driving

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We consider the problem of prediction in a highly uncertain environment.

1 Introduction

The contributions of this work are the following:

- We introduce a new open-source environment to test methods for autonomous driving, based on a large dataset of real-world driving data.
- We introduce a novel method for prediction under uncertainty which is simple to train, does not make assumptions about a prior distribution over latent variables or require sampling at training time, and is able to generate diverse predictions for hundreds of timesteps into the future.

Most methods for stochastic prediction rely on the framework of conditional Variational Autoencoders (cVAEs). VAEs assume a prior over latent variables and include a loss term measuring the KL divergence between this prior and the distribution output by the posterior network. This helps ensure that latent variables sampled from this prior at generation time are consistent with those input to the decoder network during training. Simple priors such as diagonal Gaussians are common choices, as they enable a closed-form expression for the KL divergence between prior and posterior distributions, which in turn reduces the variance of the gradients with respect to this term. A widely recognized issue with VAEs is the so-called "posterior collapse" or "over-regularization" problem, which is caused by a conflict between the prior matching term and the reconstruction term in the loss function. If the regularization term is too high, this can cause the model to ignore the latent variables altogether in order to avoid a high cost due to the KL term. If the regularization term is too low, this can cause the model to learn a latent variable distribution which does not match with the assumed prior, leading to poor generations at test time.

We propose an alternate approach which does not place any priors on the latent variable distribution and instead relies on a non-parametric or semi-parametric sampling procedure to produce new generations at test time. By removing the prior assumption, we also do away with the need for a regularization term at training time which enforces a match between prior and posterior distributions, leading to a simple and easy to optimize loss. After training, we obtain a non-parametric estimate of the distribution over latent variables by extracting a collection of latent variables from the training set itself, which we then combine with new inputs to obtain generations. Through this non-parametric sampling procedure, we naturally obtain samples which are on the manifold of latent variables. If the prior distribution over latent variables is dependent on the input, it may additionally be necessary to restrict our sampling to a limited region of the manifold. We therefore propose and empirically investigate two ways of achieving this: one through a smoothness assumption on the trajectories of latent variables along the manifold, which translates into a nearest-neighbor based sampling scheme,

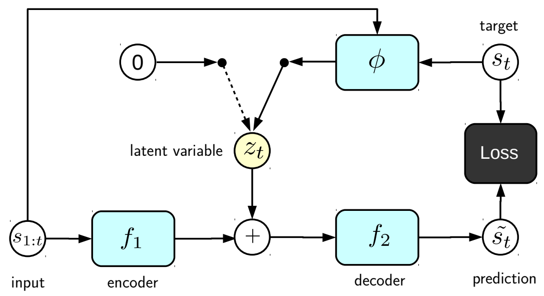


Figure 1: Prediction Model.

and by combining the non-parametric sampling scheme with a parametric likelihood model which restricts our sampling to a subregion of the manifold.

2 Prediction Model

2.1 Architecture

Our stochastic prediction model can be viewed as a conditional autoencoder paired with a non-parametric or semi-parametric sampling procedure. The architecture consists of three neural networks: an encoder f_1, a decoder f_2 and a latent variable network ϕ. For each sample, the update equations are given by:

$$\begin{aligned} z_i &= \begin{cases} 0 & \text{if } u = 1,\ u \sim \mathcal{B}(p) \\ \phi(x_i, y_i) & \text{else} \end{cases} \\ \tilde{y}_i &= f_2(f_1(x_i), z_i) \end{aligned}$$

and all networks are trained by gradient descent to optimize the following objective:

$$\begin{aligned} \mathcal{L} &= \sum_i \|y_i - \tilde{y}_i\|_2^2 && (1) \\ &= \sum_i \|y_i - f(x_i, z_i)\|_2^2 && (2) \end{aligned}$$

Note in particular that no sampling or reparamaterization is done at training time. In standard autoencoders, some mechanism is used to limit the information content of the latent representation to avoid learning trivial mappings and thus encourage the model to learn good representations. In the conditional case we are considering, we also need such a mechanism to prevent the network from simply reconstructing y_i while ignoring the input x_i. In our experiments we found setting the dimension of z_i to be of much lower dimension than y_i to work well, however other methods such as adding a sparsity or contractive penalty term could also be used.

2.2 Sampling

Our sampling sheme is based on ideas from manifold learning, for which there is an extensive literature [1, 2, 3, 4]. These methods assume we are given a set of points $\{x_i\}$ which are assumed to lie on a low-dimensional manifold $\mathcal{M}$, and approximate the manifold by forming a graph whose vertices are $\{x_i\}$ and whose edges (which can be weighted or unweighted) depend on the Euclidean distance between them. In particular, two points are connected by an edge if they are close in Euclidean distance, as this approximates the geodesic distance along the manifold as it becomes small. Paths along the manifold can then be approximated by paths along the graph, and it can be shown that given a sufficiently dense sampling of points, graph distances converge to geodesic distances [5].

Our sampling scheme is based on such ideas, in particular, we can treat the vectors z_t extracted from the training set as points sampled from the unknown latent manifold, form a graph approximation to this manifold using these points, and sample trajectories on this manifold to generate new trajectories in the original input space.

After training, we extract all vectors z_i from the training set and use these as inputs to the stochastic prediction model. We refer to this set of vectors as $\mathcal{Z}$. We explore three sampling methods:

No Prior For a given input x, we sample some z from $\mathcal{Z}$ uniformly at random. In this case the latent variable is independent of the input, similarly to the Fixed Prior VAEs used in [**?**].

KNN Prior Here we make a smoothness assumption on the trajectories followed along the latent manifold. In particular, we assume that z_{t+1} tends to be closer (on average) to z_t than a random other vector in $\mathcal{Z}$. This assumption is motivated by a smoothness assumption on the inputs: if s_t and s_{t+1} are close and ϕ is a smooth function, then z_t and z_{t+1} will be close as well. This smoothness assumption can be reflected by restricting the choice of z_{t+1} we sample from $\mathcal{Z}$ to the k nearest neighbors of z_t. This approach can be implemented by construcing a k nearest neighbor graph and performing a random walk along this graph (see algorithm 1).

Learned Prior We can also learn an input-dependent prior by training a neural network to assign likelihood values to different vectors in $\mathcal{Z}$ for a given input x. Specifically, we train a Mixture Density network [6] $\psi(x)$ which outputs the parameters $(\{\pi_j\}, \{\mu_j\}, \{\sigma_j\})$ of a Gaussian mixture model and is trained to maximize the likelihood of z_t under this model. Note that Gaussian mixture models can approximate arbitrary distributions given a sufficient number of components, hence this is not a conceptual limitation to the class of prior distributions which can be represented with our approach.

A tradeoff in modeling distributions using non-parametric approaches is that they trade flexibility for increased memory and computational cost. All the approaches above have a memory footprint which grows with the size of the dataset. In terms of computational cost, the first approach requires $\mathcal{O}(1)$ at inference time as we are simply sampling from $\mathcal{Z}$. The second approach allows us to construct the graph offline since it is input-independent, hence sampling is also $\mathcal{O}(1)$. The third approach is input-dependent, however we note that is can be cast as a nearest-neighbor problem by first sampling a point from the Gaussian mixture model output by ψ, and then mapping it to the nearest vector in $\mathcal{Z}$. Finding nearest neighbors is a well-studied problem for which fast exact or approximate solutions exist, such as kd-trees or locally sensitive hashing. In our experiments, we found that a simple linear exact search using a subset of the training data was sufficiently fast using a GPU and achieved good results.

Algorithm 1 My algorithm

```
 1: Input: Time series $\{s_1 ... s_T\}$.
 2: Train latent model:
 3: while not converged do
 4:     $z_t = f_\phi(s_{1:t}, s_{t+1})$
 5:     $\tilde{s}_{t+1} = f_\theta(s_{1:t}, z_t)$
 6:     $\ell(\theta, \phi) = \|s_{t+1} - \tilde{s}_{t+1}\|_2^2$
 7:     $\theta \leftarrow \theta - \eta \nabla \theta$
 8:     $\phi \leftarrow \phi - \eta \nabla \phi$
 9: procedure EstimateLatentManifold
10:     $V \leftarrow \{\}$
11:     $E \leftarrow \{\}$
12:     for $t = 1 : T$ do
13:         $z_t = f_\phi(s_{1:t}, s_{t+1})$
14:         $V[t] = z_t$
15:     for $t = 1 : T$ do
16:         $E[t] \leftarrow$ list of $k$ nearest neighbors of $V[t] = z_t$ in $V$
        return $G = (V, E)$
17: procedure Generate($s_0, s_1$)
18:     Initialize $z_1 = f_\phi(s_0, s_1)$
19:     for $t = 1 : T$ do
20:         $\tilde{s}_{t+1} = f_\theta(s_{1:t}, z_t)$
```

3 Related Work

In recent years, several works have explored prediction of complex time series such as video [7]. These typically train models to predict future frames with the goal of learning good representations which disentangle factor of variation and can be used for unsupervised learning [8? , 9] or learn action-conditional forward models which can be used for planning [10, 11, 12?]. Several works have included latent variables as a means to model the uncertainty, using the framework of Variational Autoencoders [13, 14]. In contrast to VAEs, our model does not place any priors on the latent variable distribution, which removes the need for an additional loss term enforcing consistency between the prior and posterior. Additionally, our approach does not require any sampling at training time which reduces the variance in the gradients.

Our method is closely related to Gated and Relational Autoencoders [15, 16], which were used to learn transformations between pairs of images in an unsupervised manner. The general architecture and loss is similar in both our works, however their focus was on representation learning for static images while ours is on video generation for use in planning.

- Video Prediction [7], stochastic: using the framework of Variational Autoencoders [17].
- Mixture Density Networks [6].
- VQ-VAE, GLO, Gated Autoencoders

4 Data sets

We trained our latent variable forward model on a synthetic data set first — where we are in partial control of the stochasticity of the data — and on real vehicle trajectories afterwards.

4.1 Synthetic data set

Variable number of lanes, vehicles, and so on.

4.2 NGSIM I-80

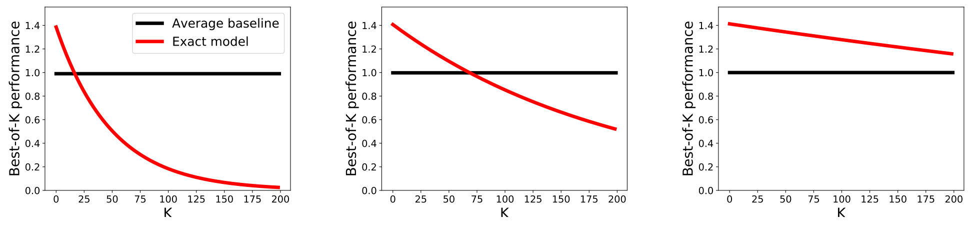
The Next Generation SIMulation program's Interstate 80 (NGSIM I-80) freeway data set [18] captures 3 segmets of 15 minutes each of a 0.5 km long section of the eastbound I–80 in the San Francisco Bay area in Emeryville, CA, in 2005. Some numbers:

- 4.5M distinct data points
- data point: 'Vehicle ID', 'Frame ID', 'Total Frames', 'Global Time', 'Local X', 'Local Y', 'Global X', 'Global Y', 'Vehicle Length', 'Vehicle Width', 'Vehicle Class', 'Vehicle Velocity', 'Vehicle Acceleration', 'Lane Identification', 'Preceding Vehicle', 'Following Vehicle', 'Spacing', 'Headway'
- 3 sections of 15 minutes each
- 2052, 1836, 1790 individual vehicles per section, for a total of 5.5k vehicles, including

5 Experiments

We evaluate our prediction model using two quantitative metrics in addition to visual evaluation.

- We follow the protocol used in previous works on stochastic generation [19, 13, 14], where we generate a fixed number of samples using the stochastic model, compare each sample to the ground truth sample in the test set, and report the error of the best match. This method provides some measure of how well the stochastic model covers the space of possible futures, although it does not necessarily reflect the realism of all the generated samples.



(a) $n = 50$ (b) $n = 200$ (c) $n = 1000$

Figure 2: Best-of-k performance of average baseline model and true model for different numbers of possible futures n. As the number of futures increases, it becomes increasingly difficult for the true model to outperform the baseline according to this metric.

5.1 A note on evaluation

We provide a discussion of evaluation metrics, and their relationship to the stochasticity of the prediction task. Here we argue that the metric used in previous work [19, 13, 14] is only meaningful when the number of samples generated is of similar magnitude to the number of possible futures. Consider the task of modeling a distribution of one-hot vectors, where each sample is a binary vector with all zeros except for a single 1, whose index is chosen uniformly between 1 and n. A baseline model which always predicts the average $\tilde{y} = (1/n, 1/n, ..., 1/n)$ (which is clearly not a valid sample in the target distribution) will have an expected loss of $\sqrt{(n-1)/n}$. A stochastic model which exactly captures the true distribution will have an expected loss of $\sqrt{2} \cdot F(k, k, (n-1)/n)$, where F refers to the cumulative distribution function of a Binomial random variable with $p = (n-1)/n$ (proofs can be found in the Appendix). Curves of both of these loss functions for different n are shown in Figure 2. These illustrate that as n becomes large, an increasingly high number of samples is required for the correct stochastic prediction model to outperform the deterministic baseline according to this metric.

The number of possible futures of a stochastic process can grow exponentially with the number of time steps. We therefore use this metric to evaluate model predictions over a limited time horizon in order to give models a chance to capture the true future with a number of samples which is computationally tractable. Since we are also interested in the the performance of models to generate predictions over long time horizons, we also provide a complementary measure where we train an separate model to distinguish between real and generated images after the stochastic model has been trained. This has been used in several works to evaluate GANs [20, 21, 22], however the approach is applicable to generative models more broadly. In this work we use the Least-Squares GAN Criterion [23], as this was found to be both stable and sensitive in [22].

- Gradient of z vectors for single timestep

5.2 Results

To test the generality of our video prediction model, we also evaluated it on two other video prediction datasets used in the literature: Stochastic Moving MNIST introduced in [14] and the BAIR robot pushing dataset originally introduced in [24] and used for video prediction in [13, 14]. We used the same model architectures as in [14]: for BAIR, a VGG16 frame encoder and decoder combined with a 2-layer LSTM network with 128 hidden units, and for SM-MNIST a DCGAN encoder and decoder also combined with a 2-layer LSTM with 128 hidden units. All models were training with Adam [25] with learning rate 0.002. The main difference is that we did not use a prior network at training time, did not include a KL term in the loss function, and instead sampled latents at test time using the approaches described in Section 2. We also set the dimension of z to be lower (2 dimensions) and trained the model with dropout parameter $p = 1$ on the latent code (corresponding to purely deterministic mode) until the loss stopped decreasing, after which we set $p = 0.5$.

Average results using the best-of-k metric along with 95% confidence intervals are shown in Figure 4. Our method performs slightly worse on average than the model of [14], however this difference is not

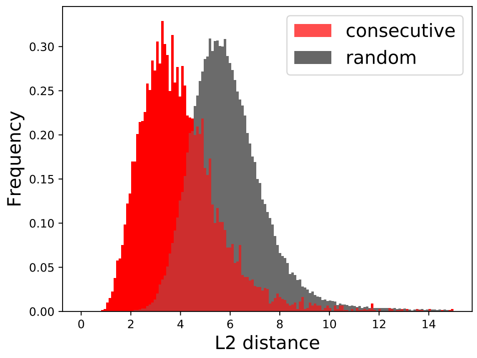


Figure 3: Distribution of distances between consecutive z and random pairs of z vectors extracted from the I-80 training set. Consecutive vectors are closer on average, which supports the smoothness assumption underlying Algorithm **??**.

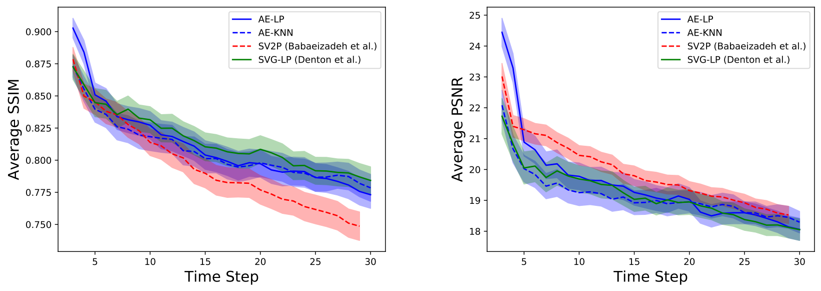


Figure 4

statistically significant. Our method and that of [14] perform better than [13] according to the SSIM metric and worse according to PSNR. Examples of video generations can be found at `url`.

6 Appendix

6.1 Proof of Expected Loss

The baseline model always produces the same prediction $\tilde{y} = (\frac{1}{n}, ..., \frac{1}{n})$. Each sample from the true distribution is of the form $y_i = (0, ..., 0, 1, 0, ..., 0)$ with a single non-zero element.

$$\begin{aligned} \mathbb{E}[\mathcal{L}_{\text{baseline}}] &= \sqrt{\sum_{i=1}^{n} (\tilde{y}_i - y_i)^2} \\ &= \sqrt{(1 - \frac{1}{n})^2 + (n-1)\left(\frac{1}{n}\right)^2} \\ &= \sqrt{\frac{(n-1)^2 + (n-1)}{n^2}} \\ &= \sqrt{\frac{n-1}{n}} \end{aligned}$$

To compute the best-of-k loss, the true stochastic model produces k samples of the same form as the one drawn from the true distribution, i.e. $\tilde{y}_k = (0, ..., 0, 1, 0, ..., 0)$, where the non-zero index is chosen uniformly from 1 to n. Let j denote the non-zero index of the sample drawn from the true distribution and $i_1, i_2, ..., i_k$ non-zero indices of the samples output by the stochastic model. The probability of each sample *not* matching y_j is equal to $p = \frac{n-1}{n}$. Therefore, the probability of none of the generated samples matching y_j is equal to the cumulative distribution function of a binomial random variable $B(k,p)$, denoted $F(k,k,p)$. In the case of a generated sample matching y_j, the best-of-k loss will equal 0, otherwise it will equal $\sqrt{2}$.

Putting this together, the expected best-of-k loss is given by:

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\text{best-of-}k}] &= \text{P(at least one sample matches)} \cdot 0 + \text{P(none of the samples match)} \cdot \sqrt{2} \\ &= \sqrt{2} \cdot F\Big(k, k, \frac{n-1}{n}\Big)\end{aligned}$$

6.2 Relationship to VAEs

Our model can also be viewed through the lens of Variational Autoencoders, as a type of conditional VAE with a zero-variance posterior network and a uniform categorical prior. In this case, the KL term reduces to a constant which can be ignored during training; details can be found in the Appendix.

Fixed Prior

$$p_\psi(z_j|x_i, y_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{3}$$

$$p_\phi(z_j|x_i) = \frac{1}{M} \tag{4}$$

$$\tag{5}$$

The KL divergence is therefore:

$$\begin{aligned}\text{KL}(p_\phi(z|x_i)||p_\psi(z|x_i, y_i)) &= \sum_j p_\psi(z_j|x_i, y_i) \log \frac{p_\psi(z_j|x_i, y_i)}{p_\phi(z_j|x_i)} && (6) \\ &= 1 \cdot \log \frac{1}{\frac{1}{M}} && (7) \\ &= \log M && (8)\end{aligned}$$

Learned Prior

$$p_\psi(z_j|x_i, y_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{9}$$

$$p_\phi(z_j|x_i) = \frac{h(z_j; x_j, \theta)}{\sum_{k=1}^{M} h(z_k; x_j, \theta)} \tag{10}$$

The KL divergence is therefore:

$$\begin{aligned}
\mathrm{KL}(p_\phi(z|x_i)||p_\psi(z|x_i,y_i)) &= \sum_j p_\psi(z_j|x_i,y_i)\log\frac{p_\psi(z_j|x_i,y_i)}{p_\phi(z_j|x_i)} && (11)\\
&= 1\cdot\log\frac{1}{\frac{h(z_j;x_j,\theta)}{\sum_{k=1}^M h(z_k;x_j,\theta)}} && (12)\\
&= -\log\frac{h(z_j;x_j,\theta)}{\sum_{k=1}^M h(z_k;x_j,\theta)} && (13)\\
&= -\log h(z_j;x_j,\theta)+\log\sum_{k=1}^M h(z_k;x_j,\theta) && (14)
\end{aligned}$$

The last term is a constant and can be ignored.

References

[1] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[2] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

[3] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

[4] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[5] Mira Bernstein, Vin De Silva, John C. Langford, and Joshua B. Tenenbaum. Graph approximations to geodesics on embedded manifolds, 2000.

[6] Christopher Bishop. Mixture density networks. Technical report, January 1994.

[7] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015.

[8] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681, 2015.

[9] Emily Denton and Vighnesh Birodkar. Unsupervised learning of disentangled representations from video. *CoRR*, abs/1705.10915, 2017.

[10] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in atari games. *CoRR*, abs/1507.08750, 2015.

[11] Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157, 2016.

[12] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016.

[13] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *CoRR*, abs/1710.11252, 2017.

[14] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *CoRR*, abs/1802.07687, 2018.

[15] Roland Memisevic and Geoffrey Hinton. Unsupervised learning of image transformations. pages 1–8, 07 2007.

[16] R. Memisevic. Learning to relate images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1829–1846, Aug 2013.

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 12 2013.

[18] John Halkias and James Colyar. NGSIM interstate 80 freeway dataset. *US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA*, 2006.

[19] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 835–851, Cham, 2016. Springer International Publishing.

[20] Ivo Danihelka, Balaji Lakshminarayanan, Benigno Uria, Daan Wierstra, and Peter Dayan. Comparison of maximum likelihood and gan-based training of real nvps. *CoRR*, abs/1705.05263, 2017.

[21] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed. Variational approaches for auto-encoding generative adversarial networks. *CoRR*, abs/1706.04987, 2017.

[22] Daniel Jiwoong Im, Alllan He Ma, Graham W. Taylor, and Kristin Branson. Quantitatively evaluating GANs with divergences proposed for training. In *International Conference on Learning Representations*, 2018.

[23] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[24] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, pages 344–356, 2017.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.