Learning a Driving Simulator

Eder Santana *
University of Florida
eder@comma.ai

George Hotz
comma.ai
george@comma.ai

Abstract

comma.ai's approach to Artificial Intelligence for self-driving cars[1] is based on an agent that learns to clone driver behaviors and plans maneuvers by simulating future events in the road. This paper illustrates one of our research approaches for driving simulation. One where we learn to simulate. Here we investigate variational autoencoders with classical and learned cost functions using generative adversarial networks for embedding road frames. Afterwards, we learn a transition model in the embedded space using action conditioned Recurrent Neural Networks. We show that our approach can keep predicting realistic looking video for several frames despite the transition model being optimized without a cost function in the pixel space.

1 Introduction

Self-driving cars are one of the most promising prospects for near term Artificial Intelligence research. The state-of-the-art of this research leverages large amounts of labeled and contextually rich data, which abounds in driving. From a complex perception and controls perspective, the technology to correctly solve driving can potentially be extended to other interesting tasks such as action recognition from videos and planning. An approach for self-driving cars that is economically attractive while still expanding the AI frontier should be based on vision, which is also the main sensor used by a human driver.

Vision based controls and reinforcement learning had recent success in the literature [7] [8] [9] [10] mostly due to (deep, recurrent) neural networks and unbounded access to world or game iteraction. Such interactions provide the possibility to revisit states with new policies and to simulate future events for training deep neural network based controllers. For example, Alpha Go [9] used deep convolutional neural networks to predict the probability of winning the game from a given state using the outcomes of several games, lots of which were playing but previous iterations of Alpha Go itself. The Go game engine was also used for simulating possible unfoldings of the game and Markov Chain Tree Search. For learning to play Torcs [7] or Atari [8] using only the game screen, it was also necessary to use several hours of game play.

For the controls problems where unbounded iteraction of a learning agent with the world is not easily affordable there are two possible solutions. The solutions are to either hand code a simulator or to learn to simulate. Hand coding a simulator involves domain expertise for defining rules of physics and modelling the randomness of the world. But for most cases, such expertise already encompass all the information for posing the related controls problem, for example flight simulators [11], robot walking [12], etc.

Here instead we focus on learning to simulate aspects of the world from examples of a human agent. We focus on generating video streams of a front facing camera mounted in the car windshield.

*This work was done during a Summer 2016 internship at comma.ai. The companion material of this paper can be found online: http://research.comma.ai

[1] https://www.youtube.com/watch?v=KTrgRYa2wbI

Figure 1: 80×160 samples from the driving dataset.

Previous successful approaches for learning to simulate for controls were based on state space representations of the agent physics [13]. Other vision only models focused on low dimensional [15] or videos with simple textures, such as those of Atari games [14] [16]. Texture rich approaches were based on passive video prediction for action recognition [17] without the possibility of an external agent modifying the future of the video and without generating a compact representations of the data.

This paper contributes to this learned video simulation literature. No graphics engine or world model assumptions are made. We show that it is possible to train a model to do realistic looking video predictions, while calculating a low dimensional compact representation and action conditioned transitions. We leave controls on the simulated world for future work. In the next section we describe the dataset we used to study video prediction of real world road videos.

2 Dataset

We are publicly releasing part of the driving data used in this paper. The dataset has the same video and sensors used in the `comma.ai` self-driving car test platform.

We mounted a Point Grey camera in the windshield of an Acura ILX 2016, capturing pictures of the road at 20 Hz. In the released dataset there is a total of 7.25 hours of driving data divided in 11 videos. The released video frames are 160×320 pixels regions from the middle of the captured screen. Beyond video, the dataset also has several sensors that were measured in different frequencies and interpolated to 100Hz. Example data coming from sensors are the car speed, steering angle, GPS, gyroscope, IMU, etc. Further detail about the dataset and measurement equipment can be found online in the companion website.

We also record the time stamps at which these sensors were measured and the time stamps the camera frames were captured. In our production pipeline the measurement times and simple linear interpolation are used to sync the sensors and camera frames. We are releasing the raw sensors and camera frames in HDF5 files for easy to use in machine learning and controls software.

In this paper we focus on the camera frames, steering angle and speed. We preprocessed the camera frames by downsampling them to 80×160 and renormalizing the pixel values between -1 and 1. Sample images of this transformed dataset are shown in Fig. 1.

In the next section we define the problem investigated in this paper.

3 Problem definition

Let x_t denote the t-th frame of the dataset. $X_t = \{x_{t-n+1}, x_{t-n+2}, ..., x_t\}$ denotes a n frames long video. These frame are associated with control signals $S_t = \{s_{t-n+1}, s_{t-n+2}, ..., s_t\}$, $A_t = \{a_{t-n+1}, a_{t-n+2}, ..., a_t\}$ corresponding to the ego speeds and steering angles of the car. Learning to simulate the road can be defined as estimating the function $F : \mathbb{R}^{80\times160\times3\times n} \times \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^{80\times160\times3}$ that predicts $x_{t+1} = F(X_t, S_t, A_t)$.

Note that this definition is high dimensional and the dimensions are highly correlated. In Machine Learning problems like those are known to converge slowly or underfit the data [26]. For example, previous work [20] showed that when using convolutional dynamic networks without proper regularization, the model simply learns a transformation close to the identity while still getting low cost scores due to correlations in time. Previous approaches learned the function F directly, but were demonstrated only in very simple, artificial videos [14]. Recently, it was shown [20] [17] that it is possible to generate videos with complex textures, but the referred work did not address the problem of action conditioned transitions and they do not generate a compact intermediate representation of the data. In other words, their models are fully convolutional with no downsampling and no low dimensional hidden codes. We believe that compact intermediate representations are important for our research because probabilities, filtering, and consequently controls, are ill defined in very high dimensional dense spaces [18].

As far as we know, this is the first paper that tries to learn video predictions from real world highway scenes. As such, in this first paper, we decided to learn the function F in a piecewise manner, so we could debug its parts separately. First we learned an autoencoder to embed the frames x_t into a Gaussian latent space $z_t \in \mathbb{R}^{2048}$, where the dimensionality 2048 was chosen experimentally and the Gaussian assumption enforced with variational autoencoding Bayes [1]. This first step simplified the problem from learning transitions directly in the pixel space to learning in the latent space. Beyond that, assuming that an autoencoder is correctly learned respecting the latent space Gaussianity, we can generate realistic looking videos as long as the transition model never leaves the high density region of the embedding space. This high density region is a hypersphere of radius ρ, which in its turn is function of the dimensionality of embedding space and variance of Gaussian prior. We go into the details of the autoencoder and the transition model in the next section.

4 Driving simulator

Due to the problem complexity we decided to learn video prediction with separable networks. We leave investigations of the end-to-end approach for future work. The proposed architecture is based on two models: an autoenecoder for dimensionality reduction and an action conditioned RNN for learning the transitions. Our complete model is depicted in Fig. 2.

4.1 Autoencoder

To learn the data embedding we chose a model that enforced the probability distribution of the latent space to be Gaussian. Specially to avoid discontinuous regions of low probability inside a hypersphere centered in the origin. Such discontinuities could make it harder to learn a continuous transition model in the latent space. Variational autoencoders [1] and related work [19] [21] had recent sucess learning generative models with Gaussian priors in the latent and original data space. Unfortunately, the Gaussian assumption in the original data space does not hold for natural images and VAE predictions usually look blurry (see Fig. 3). On the other hand, generative adversarial networks (GAN) [22] and related work [2] [3] learn the generative model cost function alongside the generator. They do so by alternating the training of generative and discriminator networks. The generative model transforms samples from the latent space distribution into data from a desired dataset. The discriminator network tries to tell samples from the desired dataset from images sampled by the generator. The generator is trained to "fool" the discriminator, thus the discriminator can be considered a learned cost function for the generator.

For our purposes we need to learn not only the generator from the latent to the road images space. But also an encoder from road images back to the latent space. We need to combine VAE and GANs. Intuitively, one could simply combine the VAE approach with a learned cost function. In

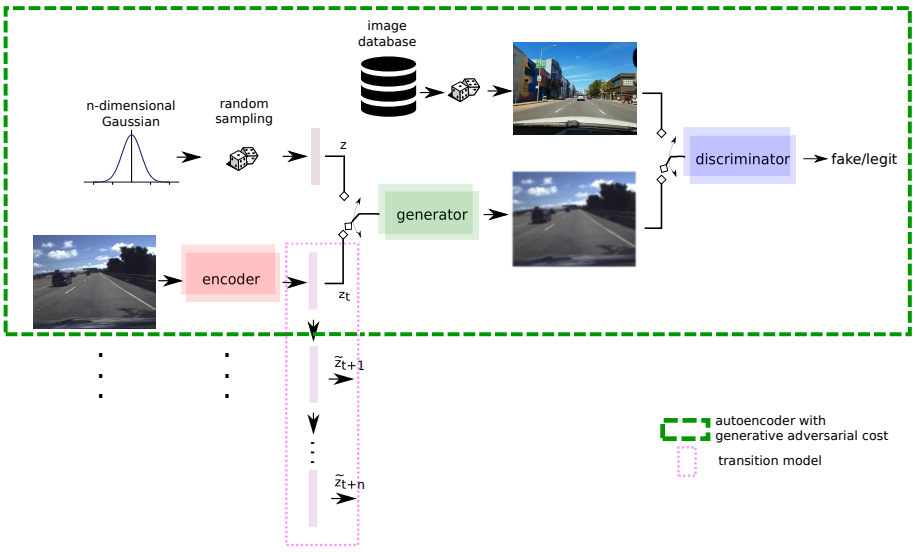

Figure 2: Driving simulator model: an autoencoder trained with generative adversarial costs coupled with a recurrent neural network transition model

the literature Donahue et. al [23] proposed bidirectional GANs that learns both generation and encoding as bijective transformations. Lamb et. al. [24] proposed discriminative generative networks that uses distances between features of a previously trained classifier as part of the cost function. Finally, Larsen et. al. [25] proposed to train a VAE alongside a GAN, where the encoder optimizes both the Gaussian prior in the latent space and the similarity between features extracted by the GAN discriminator network. The generator receives as inputs random samples from the latent space distribution and outputs of the encoder network. The generator is optimized to fool the discriminator and minimize the similarity between original and decoded images. The discriminator is still only trained to tell fake from legit images.

We used Larsen et. al. [25] approach to train our autoencoder. In Fig. 2 we show the schematic diagram of this model as part of our architecture. According to [25] the encoder (Enc), generator (Gen) and discriminator (Dis) networks are optimized to minimize the following cost function:

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}. \tag{1}$$

In (1) $\mathcal{L}_{prior} = D_{KL}(q(z|x)||p(z))$ is the Kullback-Liebler divergence between the distribution of the encoder outputs, $q(z|x)$, and a prior distribution, $p(z)$. This is the same VAE regularizer. Here the prior distribution is $p(z)$ is a Gaussian $\mathcal{N}(0, 1)$ and we use the *reparametrization trick* [1] to optimize that regularizer. Thus, during training we have $z = \mu + \epsilon\sigma$ and at test time $z = \mu$, where μ and σ are outputs of the encoder network and ϵ is a Gaussian random vector with the same number of dimensions as μ and σ.

$\mathcal{L}_{llike}^{Dis_l}$ is an error calculated using Dis_l, the hidden activations of the l-th layer of the discriminator network, Dis. The activations are calculated using a legit image x and its corresponding encoded-decoded version $Gen(Dis(x))$. Assuming $y_l = Dis_l(x)$ and $\tilde{y}_l = Dis_l(Gen(Enc(x)))$, we have $\mathcal{L}_{llike}^{Dis_l} = \mathbb{E}\left[(y_l - \tilde{y}_l)^2\right]$. During training, Dis is considered constant with respect to this cost to avoid trivial solutions.

Finally, $\mathcal{L}_{GAN}$ is the generative adversarial network cost [22]. That cost function represents the *game* between Gen and Dis. When training Dis, both Enc and Gen are kept fixed and we have

$$\mathcal{L}_{GAN}^{Dis} = log(Dis(x)) + log(1 - Dis(Gen(u))) + log(1 - Dis(Gen(Enc(x))), \tag{2}$$

where $u \sim \mathcal{N}(0, 1)$ is another random variable. The first r.h.t of (2) represents the log-likelihood of Dis recognizing legit samples and the other two terms represents its log-likelihood to tell fake samples generated from both random vectors u or codes $z = Enc(x)$. When training Gen, we keep

both Dis and Enc fixed and we have

$$\mathcal{L}_{GAN}^{Gen} = log(Dis(Gen(u))) + log(Dis(Gen(Enc(x))), \tag{3}$$

which accounts for Gen being able to fool Dis. Following [25], the gradient of $\mathcal{L}_{GAN}$ w.r.t Enc is considered zero during training.

We trained this autoencoder for 200 epochs. Each epoch consisted of 10000 gradient updates with a batch size of 64. Batches were sampled randomly from driving data as described in the previous section. We used Adam for optimization [4]. The autoencoder architecture followed Radford et. al. [3], with the generator made of 4 deconvolutional layers each one followed by batch normalization and leaky-ReLU activations. The discriminator and encoder consisted of convolutional layers where each but the first layer was followed by batch normalization. The activation function was ReLU. Dis_l is the output of the 3rd convolutional layer of the decoder, before batch normalization and ReLU are applied. The output size of the discriminator network is 1 and its cost function was binary cross-entropy. The output size of the encoder network is 2048. This compact representation is almost 16 times smallet than the original data dimensionality. For network sizes and further details, please check Fig. 2 and the companion code of this paper. Sample encoded-decoded and target images are shown in Fig. 3.

After training the autoencoder, we fix all its weights and use Enc as preprocessing step for training the transition model. We describe the transition model in the next section.

4.2 Transition model

After training the autoencoder described above we obtain the transformed dataset applying Enc : $x_t \mapsto z_t$. We train $RNN : z_t, h_t, c_t \mapsto z_{t+1}$ to represent the transitions in the code space:

$$\begin{aligned} h_{t+1} &= tanh\left(Wh_t + Vz_t + Uc_t\right), \\ \tilde{z}_{t+1} &= Ah_{t+1} \end{aligned} \tag{4}$$

where W, V, U, A are trainable weights, h_t is the hidden state of the RNN and c_t are the concatenated control speed and angle signal. We leave LSTMs, GRU and multiplicative iterations between c_t and z_t for future investigations. The cost function for optimizing the trainable weights is simply the mean square error:

$$\mathcal{L}_{RNN} = \mathbb{E}\left[(z_{t+1} - \tilde{z}_{t+1})^2\right]. \tag{5}$$

It is easy to see that (5) is optimal because we impose a Gaussian constraint $\mathcal{L}_{prior}$ in the distribution of the codes z when training the autoencoder. In other words, MSE equals up to a constant factor the log-likelihood of a normally distributed random variable. Given a predicted code $\tilde{z}_{t+1}$ we can estimate future frames as $\tilde{x}_{t+1} = Gen(\tilde{z}_{t+1})$.

We trained a transition model with sequences of length of 15 frames. We used teacher forcing in the first 5 frames and fed the outputs back as new inputs in the remaining 10 frames. In other words, $z_1, ..., z_5$ were calculated using $Enc(x_t)$ and fed as input. We fed the outputs $\tilde{z}_6, ..., \tilde{z}_{15}$ back as input. In the RNN literature feeding outputs back as input is informally referred as RNN *hallucination*. It is important to note that we consider the gradient of the outputs equal to zero when they are fed back as inputs to avoid trivial solutions.

5 Results

We spent most of the effort of this research investigating autoencoding architectures that could preserve the road texture. As mentioned before we studied different cost functions. Although results were similar in terms of MSE, a learned cost function using generative adversarial networks had the most visually appealing results. In Fig. 3 we show decoded images with models trained using two different cost functions. As expected, MSE based neural networks generate blurred images. The main issues for our purposes is that blurring connects the lane marks into a single long lane. Also

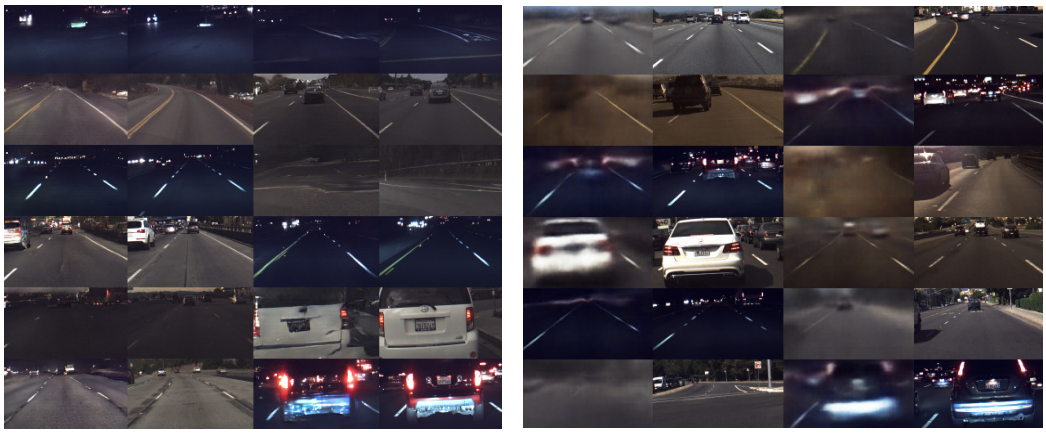

Figure 3: Samples using similar fully convolutional autoencoders. Odd columns show decoded images, even columns show target images. Models were trained using (a) generative adversarial networks cost function (b) mean square error. Both models have MSE in the order of 10^{-2} and PSNR in the order of 10.

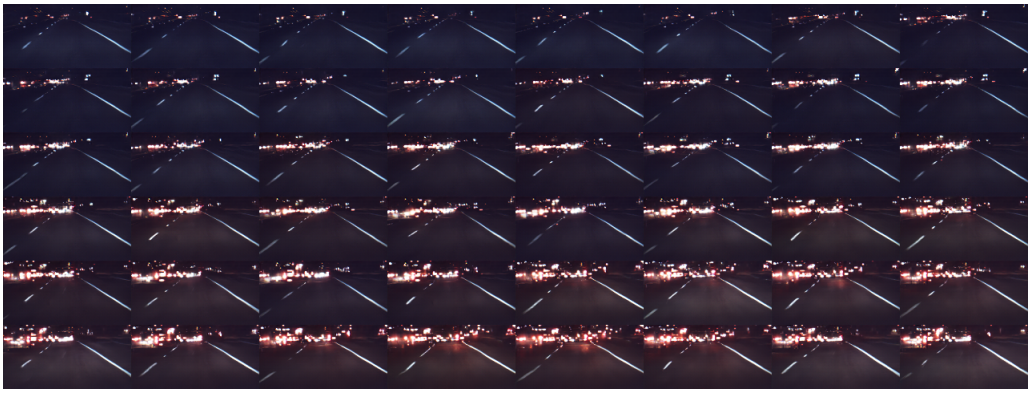

Figure 4: Samples generated by letting the transition model *hallucinate* and decoding $\tilde{z}$ using Gen. Note that Gen was not optimized to make these samples realistic, which support our assumption that the transition model did not leave the code space.

blurring reconstruction does not preserve leading car edges. This is not desired for several reasons, the main one is that it gets rid of one of the main clues used for visual odometry and leading car distance estimation. On the other hand MSE learns to generate curved lane markings faster than the adversarial based model. It is possible that learning to encode pixels with the car steering angle information should avoid this problem. We leave this investigation for future work.

Once we obtained a good autoencoder, we trained the transition model. Results of predicted frames are shown in Fig. 4 . We trained the transition model in 5Hz videos. The learned transition model keeps the road structure even after 100 frames. When sampling from the transition model with different seed frames we observed simulations of common driving events such as passing lanes, approaching leading cars and leading cars moving away. The model failed to simulate curves. When we initialized the transition model with frames of driving on a curve, the transition model quickly straightens the lanes and goes back to simulate *going forward*. Nevertheless, it is promissing that we could learn transitions for video generation without explicitly optimizing a cost function in the pixel space. Also, we believe that even more realistic simulations will be possible with more powerful transition models (deep RNNs, LSTM, GRU) and contextual encoding (sensor fusion of pixels plus steering angle and speed). The dataset we are releasing with this paper has all the sensors necessary to experiment with this approach.

6 Conclusions

This paper presented `comma.ai` initial research on learning a driving simulator. We investigated video prediction models based on autoencoders and RNNs. Instead of learning everything end-to-end here we first trained the autoencoder with generative adversarial network based cost functions to generate realistic looking images of the road. We followed up training an RNN transition model in the embedded space. Results from both autoencoder and transition model look realistic but more research is needed to successfully simulate all the relevant events for driving. To stimulate further research on this subject we are also releasing a driving dataset with video and several sensors such as car speed, steering angle, etc. We also release the code to train the networks we investigated here.

References

[1] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[2] Emily L Denton, Soumith Chintala, Rob Fergus, et al., "Deep generative image models using laplacian pyramid of adversarial networks," in *Advances in Neural Information Processing Systems*, 2015.

[3] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434*, 2015.

[4] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014.

[6] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[7] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al., "Human-level control through deep reinforcement learning," *Nature*, 2015.

[9] David Silver, Aja Huang, Chris Maddison, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016.

[10] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," *arXiv preprint arXiv:1603.02199*, 2016.

[11] Brian L Stevens, Frank L Lewis and Eric N Johnson, "Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems," *John Wiley & Sons*, 2015.

[12] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, et al., "Feedback control of dynamic bipedal robot locomotion," *CRC press*, 2007.

[13] HJ Kim, Michael I Jordan, Shankar Sastry, Andrew Y Ng, "Autonomous helicopter flight via reinforcement learning," *Advances in neural information processing systems*, 2003.

[14] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, et al., "Action-conditional video prediction using deep networks in atari games," *Advances in Neural Information Processing Systems*, 2015.

[15] Manuel Watter, Jost Springenberg, Joschka Boedecker and Martin Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *Advances in Neural Information Processing Systems*, 2015.

[16] Jürgen Schmidhuber, "On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models," *arXiv preprint arXiv:1511.09249*, 2015.

[17] Michael Mathieu, Camille Couprie and Yann LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[18] Ramon van Handel, "Probability in high dimension," *DTIC Document*, 2014.

[19] Eder Santana, Matthew Emigh and Jose C Principe, "Information Theoretic-Learning Auto-Encoder," *arXiv preprint arXiv:1603.06653*, 2016.

[20] Eder Santana, Matthew Emigh and Jose C Principe, "Exploiting Spatio-Temporal Dynamics for Deep Predictive Coding," *Under Review*, 2016.

[21] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly and Ian Goodfellow, "Adversarial autoencoders", *arXiv preprint arXiv:1511.05644*, 2015.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al., "Generative adversarial nets," *Advances in Neural Information Processing Systems*, 2014.

[23] Jeff Donahue, Philipp Krähenbühl and Trevor Darrell, "Adversarial Feature Learning," *arXiv preprint arXiv:1605.09782*, 2016.

[24] Alex Lamb, Vincent Dumoulin Vincent and Aaron Courville, "Discriminative Regularization for Generative Models," *arXiv preprint arXiv:1602.03220*, 2016.

[25] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle and Ole Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv preprint arXiv:1512.09300*, 2015.

[26] Jose C Principe, Neil R Euliano, W Cur Lefebvre, "Neural and adaptive systems: fundamentals through simulations with CD-ROM" *John Wiley & Sons, Inc.*, 1999.