

Using Reinforcement Learning to Learn Input Viewpoints for Scene Representation

Kevin Chiang



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-38

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-38.html>

May 14, 2019

Copyright © 2019, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Using Reinforcement Learning to Learn Input Viewpoints for Scene Representation

by

Kevin Chiang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the



University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Sergey Levine

Spring 2019

The thesis of Kevin Chiang, titled Using Reinforcement Learning to Learn Input Viewpoints for Scene Representation, is approved:

Chair		Date	<u>5/9/2019</u>
		Date	<u>5/10/2019</u>

University of California, Berkeley

Using Reinforcement Learning to Learn Input Viewpoints for Scene Representation

Copyright 2019
by
Kevin Chiang

Abstract

Using Reinforcement Learning to Learn Input Viewpoints for Scene Representation

by

Kevin Chiang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

Scene representation, the process of converting visual data into efficient, accurate features, is essential for the development of general robot intelligence. This task is drawn from human experience, as humans generally take in a novel scene by indicating important features and objects in the scene before planning their actions around these features. Recently, the Generative Query Network (GQN) was developed, which takes in random viewpoints of a scene, constructs an internal representation, and uses the representation to predict the image from an arbitrary viewpoint of the scene. GQNs have shown that it is possible to learn accurate representations of various scenes without human labels or prior domain knowledge, but one limiting factor that remains is the fact that the input viewpoints are chosen randomly. By training an agent to learn where to capture the input observations, we can supply the GQN with more useful and unique data. We show that an agent can learn through reinforcement learning (RL) to select input viewpoints that provide much more useful information than random inputs, leading to better representations, and thus more complete reconstructions, which may lead to improvements in tasks with complex environments.

To my family for their continuous support.

Contents

Contents	ii
List of Figures	iii
List of Tables	v
1 Introduction	1
2 Related Work	2
2.1 Scene Representation	2
2.2 GQN	3
2.3 Reinforcement Learning	5
3 Environments	6
4 Proposed Method	10
4.1 GQN Details	12
4.2 Reinforcement Learning Agent Details	12
5 Experiments	16
5.1 Baseline for Comparison	16
5.2 Results	17
6 Conclusion and Future Work	24
7 References	25

List of Figures

2.1	This figure is adapted from [1]. The inputs to the representation network f are the observations, and f outputs the scene representation r . r is passed into the generation network g along with the query viewpoint and latent variable and iteratively generates the predicted view.	4
3.1	Example scenes in the rooms environment. The ground truth image is the rendering of the scene from the query viewpoint.	7
3.2	Distribution of the number of objects the test scenes.	8
3.3	Distribution of the type of objects in the test scenes.	8
3.4	Distribution of the color of the walls in the test scenes.	9
3.5	Distribution of the color of the floor in the test scenes.	9
4.1	The action space of the restricted camera and free camera environments.	11
4.2	The representation network f used for the GQN. This figure is adapted from [1].	13
4.3	The generation network g used for the GQN, which is a modified version of DRAW. This figure is adapted from [1].	14
4.4	The reinforcement learning network and overall pipeline. The CNN takes a $64 \times 64 \times 3$ image as input. The first convolutional layer contains $32 \ 8 \times 8$ filters with stride 4, the second layer contains $64 \ 4 \times 4$ filters with stride 2, the third layer contains $64 \ 3 \times 3$ filters with stride 1, and the last layer is a fully connected layer with 512 units. This is then passed into the LSTM network, which has a hidden state size of 128. We use the ReLU activation function.	15
5.1	Examples of the viewpoints chosen with the baseline.	16
5.2	Training curves.	18
5.3	Comparison of GQN versus GQN+RL input viewpoints and image reconstructions in the free environment. Ground truth refers to the rendered image at the query viewpoint. GQN+RL and GQN predict are the respective models' generated images at the query viewpoint.	20
5.4	The distribution of the difference between angles of the RL chosen viewpoints when only 2 observations are allowed (10 degree bins) in the unrestricted environment. The black line indicates the distribution of random observations.	21

5.5	The distribution of the distance between the randomly chosen viewpoints when only 2 observations are allowed in the unrestricted environment.	22
5.6	The distribution of the distance between the RL chosen viewpoints when only 2 observations are allowed in the unrestricted environment.	23

List of Tables

5.1	Comparison of average reconstruction errors.	19
-----	--	----

Acknowledgments

I would first like to sincerely thank my research advisor Professor Pieter Abbeel for the opportunity to work in the BAIR lab and advising me during my candidacy. The fascinating research opportunities and invaluable support and advice he provided me since my undergraduate years have made my academic experience at UC Berkeley unforgettable. Professor Abbeel's insights have guided me through multiple projects, and much of what I know about machine learning would not have been possible without him. I would also like to thank Professor Sergey Levine for his help on my work.

I would especially like to thank my mentor Josh Tobin for supporting me throughout my research experience. He patiently and thoroughly taught me the fundamentals of meticulous research despite his busy schedule. Josh provided me with many amazing learning opportunities both at BAIR and at OpenAI, which I am extremely thankful for.

Finally, I would like to thank my friends and family for their support throughout my last four years at Berkeley.

Chapter 1

Introduction

In the growing field of AI, designing an agent around a proposed environment is a major starting point, guiding work in reducing error and improving performance. However in real world scenarios, the testing environment is seldom fully captured in the training environment. The distributional shift can lead to significant differences in the expectations of the agent and reality, eventually causing the agent to end up in an off-policy situation and perform poorly. Traditionally, attempts have been made to improve and expand the training dataset to include as much information as possible with respect to the environment, but the bottleneck is often accurately labelling the collected data, especially in complex tasks that have no clear or easy methods to label the data. While virtual environments can generate limitless amounts of training examples, collecting and labeling real data can be a tedious and high cost endeavor [2-6]. Either improving simulated environments to match their real world counterparts or collecting more real world data has a significant time and economic cost associated, with much of the time spent on the proper labeling of data. In addition, human mislabeling can lead to distributional errors that propagate to sub-optimal actions and poor performance.

The GQN (Generative Query Network) [1] attempts to solve this by removing the need for manual labeling, instead learning the latent features of the environment on its own in a self-supervised fashion. To achieve this, the network attempts to predict the scene from an arbitrary viewpoint, so the ground truth images are easily obtainable. Now the agent is able to internalize a representation of the environment without requiring a traditionally labeled dataset complemented by depth data. Furthermore, the learned representation is found to be view-independent, so we are able to capture important properties of each scene such as relative positions of objects and their textures, rather than potentially unimportant and misleading subliminal patterns in the scene.

However, one shortcoming of GQNs is that their inputs are randomly selected, which leads to the possibility that many of the images are not useful in understanding the scene and a poor representation is constructed. Thus, we propose to add an additional module that allows the agent to choose the input viewpoints that will maximally improve its conceptualization of the environment.

Chapter 2

Related Work

2.1 Scene Representation

Scene representation embodies a variety of extremely useful computer vision tasks. Object recognition tasks such as object detection [7-10], object segmentation [42], and object pose estimation [43] can theoretically be solved with scene representation techniques, as understanding a scene necessarily requires knowing which objects are in the scene and where they are located. Other tasks also solvable with scene representation include depth estimation [11, 12], semantic scene completion [13], and view selection [14].

There have also been many prior works on scene representation itself. However, traditional structure from depth and motion and multi-view geometry techniques [15-20] dictate how the 3D structure must be represented (usually as point clouds), leading to potential sub-optimal representations of some or all of the following details: textures, objects, positions, and lighting.

One particularly successful approach is simultaneous localization and mapping (SLAM) [48]. The algorithm concurrently constructs a model of the environment (a map) while estimating the state of the agent within it. The constructed map is oftentimes used as a 3D representation for aspects of interest. SLAM systems usually contain two components: one to convert raw sensor data into estimable models, and one to perform inference on the converted data from the first component. There are various implementations of these two components, but many open problems continue to plague today's SLAM algorithms, including consistent semantic-metric fusion and ignorance, awareness, and adaptation of the agent.

Other deep learning approaches, such as auto-encoders and density models [21-26] have been employed on scene representation tasks, with the expectation that their compression techniques are able to encode the 3D structure of a scene given a few viewpoints. However, there is no incentive in these approaches to relate different views of the same scene to each other, instead relying on subtle and unreliable patterns in textures to construct relationships. Approaches such as viewpoint transformation networks can learn how different viewpoints

relate to each other [18, 27], but they are extremely limited in scope and not generalizable.

2.2 GQN

GQNs [1] are able to learn scene structure from a sparse set of viewpoints. The GQN is able to produce a high level latent representation of the scene with its representation network f and predict observations with its generation network g . DeepMind proposed this network architecture based on the idea that visual learning in nature is seldom as well-supervised as required for machine learning. Labeling image segmentation, classification, or even individual pixels is unheard of. The GQN is able to learn the view-independent latent structure of each scene with only a few input observations. DeepMind refers to this as the agent’s ability to learn about the environment unaided, with only its own sensors.

Specifically, the GQN is trained and tested on a 3D environment i , where K images are collected from 2D viewpoints, represented by x_i^k and v_i^k ; these viewpoints and the corresponding images are collectively called observations $o_i = \{(x_i^k, v_i^k)\}_{k=1, \dots, K}$. The observations are passed into f , which produces a neural scene representation r . r is able to contain the scene structure, where each new observation improves the confidence in predicting reconstruction images. Reconstruction is performed by querying g with any arbitrary viewpoint v^q . The generation network takes r and stochastic latent variables z (for variability in the output) to predict the output from v^q (Figure 2.1). f and g are jointly trained, with a standard variational approximation loss function, which can be decomposed into a reconstruction likelihood term and a KL regularization term.

Formally, $r = f_\theta(o_i)$, g defines the probability density $g_\theta(\mathbf{x}|\mathbf{v}^q, \mathbf{r}) = \int g_\theta(\mathbf{x}, \mathbf{z}|\mathbf{v}^q, \mathbf{r})d\mathbf{z}$, and θ defines the learnable parameters.

In some basic examples, the GQN only needs to view a single image of a scene to produce a reasonably confident representation, on par with human-level performance. The viewpoint is selected to provide the network with the most information possible for a single view, which allows the model to perform optimally. In cases where the scene is complex enough to require multiple viewpoints, the viewpoints are either taken randomly or along a set interval, neither of which usually result in optimal viewpoints.

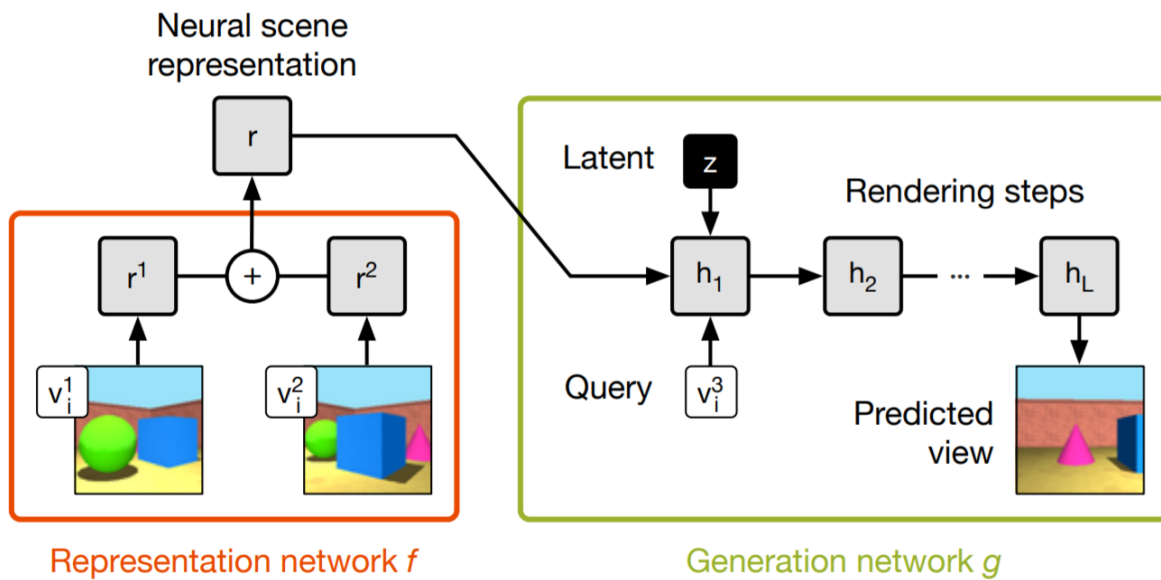


Figure 2.1: This figure is adapted from [1]. The inputs to the representation network f are the observations, and f outputs the scene representation r . r is passed into the generation network g along with the query viewpoint and latent variable and iteratively generates the predicted view.

2.3 Reinforcement Learning

In [28], the authors use the model-free reinforcement learning form of policy gradients, which we deploy in our extension. Instead of the traditional Q-value-based forms like the Bellman Equations [44], policy gradients bypass learning the predicted value of a state, instead directly predicting optimal actions. This can be more computationally efficient for large or high dimensional action spaces, where there are an unreasonable number of Q-values to enumerate. A policy-focused algorithm also has the benefit over traditional Q-learning of stochasticity. The policy can output a probability distribution over actions which allows an agent to handle perception aliasing as well as the exploration vs. exploitation trade-off. However, policy gradients tend to get stuck at local maxima or plateaus, unlike Q-learning. This can limit performance or exaggerate training times depending on the requirements of the problem.

The policy gradient algorithm starts with a policy $\pi_\theta(a|s) = P[a|s]$, where its performance is measured by score function $J(\theta) = E_{\pi_\theta}[\sum_\tau \gamma^i r_i]$. Applying gradient ascent on this score function with respect to the policy parameters θ will improve the expected discounted reward over trajectories.

Many improved policy gradient algorithms have been developed, including ACER, DDPG, PPO, and TRPO [36, 49-51], which attempt to improve the data efficiency and robustness of the algorithm, or reduce the complexity of alternative algorithms.

Chapter 3

Environments

To test our model, we created a simulated 3D environment in MuJoCo [29] similar to the ‘rooms’ environment created by DeepMind [1]. We use OpenAI Gym [30] so that an agent can move around in each scene and capture observations from any viewpoint in our environment.

Each scene is a 7×7 unit room with geometric shapes placed in the room. There are 7 possible objects (box, sphere, cylinder, cone, capsule, icosahedron, and triangle), of which between 1 to 3 of them are chosen and placed uniformly randomly in the middle 3×3 of the room; the yaw and dimensions of each object are also sampled uniformly randomly. It is possible for the objects to intersect each other. The textures of each object are randomized uniformly in HSV space, where the hue is between $[0, 1]$, the saturation is between $[0.75, 1]$, and the value is always 1.

The wall texture is sampled from 5 different options: red, green, cerise, orange, and yellow. The floor texture is sampled from 3 different options: yellow, blue, and white. A light is set 15 units above the floor, and the xy position is sampled uniformly in a 8×8 square centered at the center of the room.

We render the observation images with MuJoCo’s default OpenGL renderer, and examples are shown in Figure 3.1. We experiment with two different versions of our environment. In the simpler environment, we restrict all viewpoints to be some point in a circle of radius 2.5 units centered at the center of the room, pointing towards the center of the room. The agent picks a location on the edge of this circle by sampling between $[0, 1)$ and mapping this to an angle around the circle from a fixed point and pointing the camera towards the center of the circle. This ensures that all observations are pointing towards the objects in each scene. In the regular environment, we remove this restriction, allowing the agent to pick any viewpoint and any angle. The agent picks two points within the room; the first point determines the agent’s position, and the second point determines which direction the agent faces.

We sample 200,000 scenes each time for training. Each scene contains K observations, which is a random integer between $[1, 5]$, and a query image, which is used as the ground truth image. We also sampled and saved 20,000 scenes to be used as a test set across all experiments. Figures 3.2, 3.3, 3.4, and 3.5 reveal statistics of the test dataset.

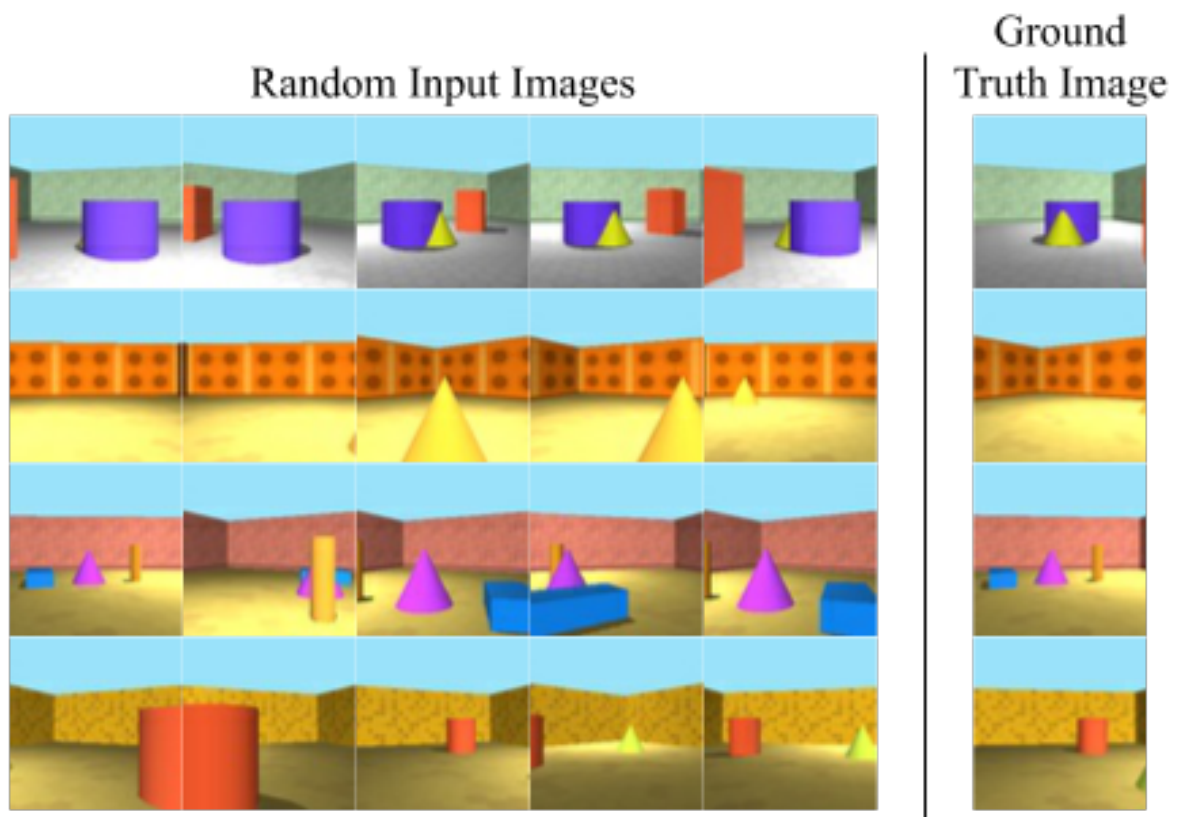


Figure 3.1: Example scenes in the rooms environment. The ground truth image is the rendering of the scene from the query viewpoint.

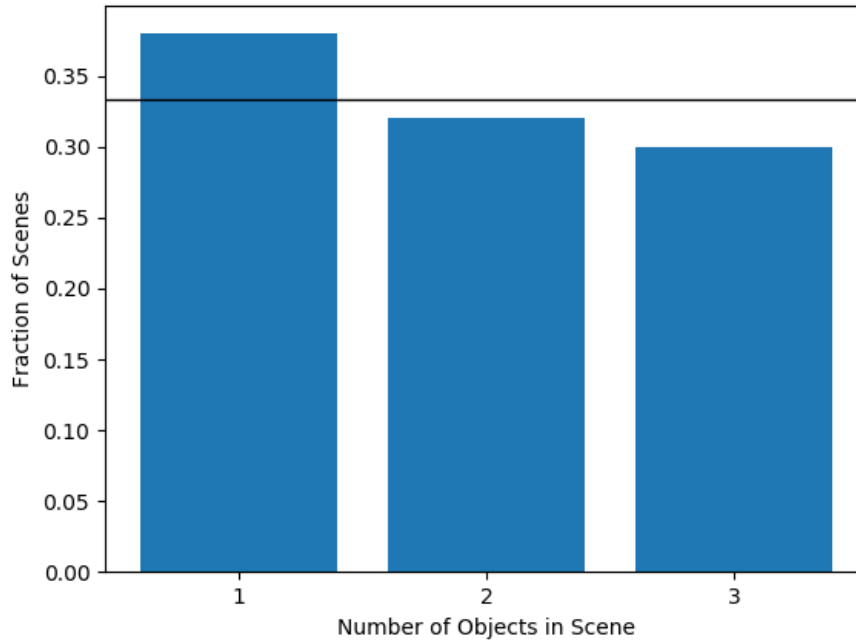


Figure 3.2: Distribution of the number of objects the test scenes.

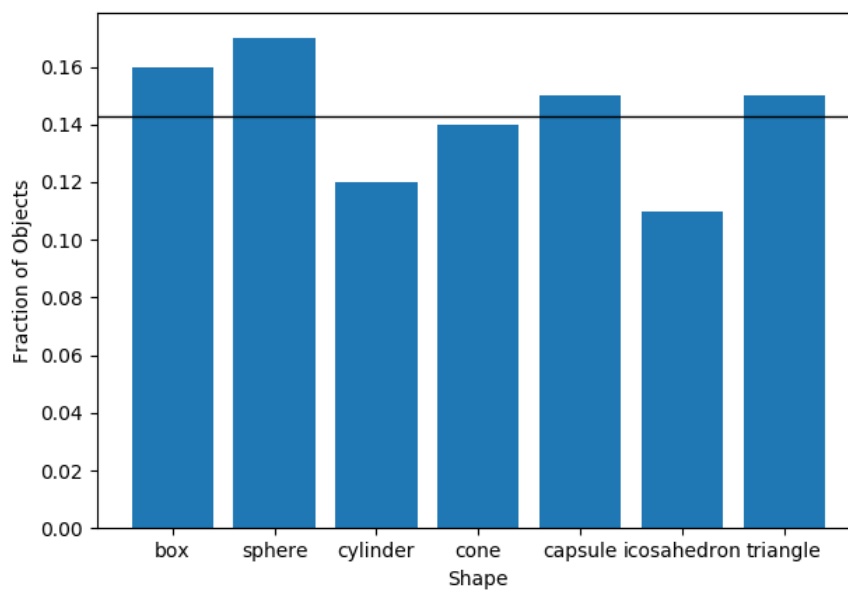


Figure 3.3: Distribution of the type of objects in the test scenes.

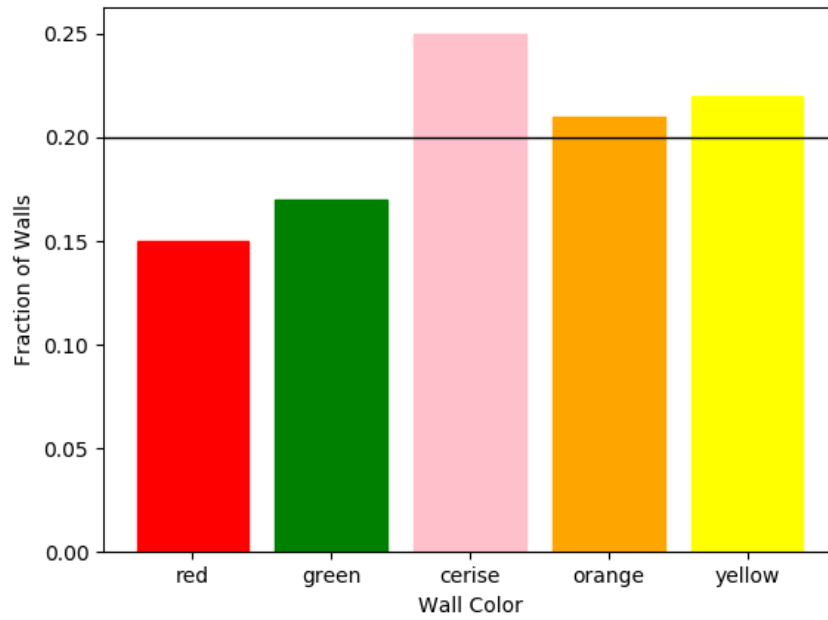


Figure 3.4: Distribution of the color of the walls in the test scenes.

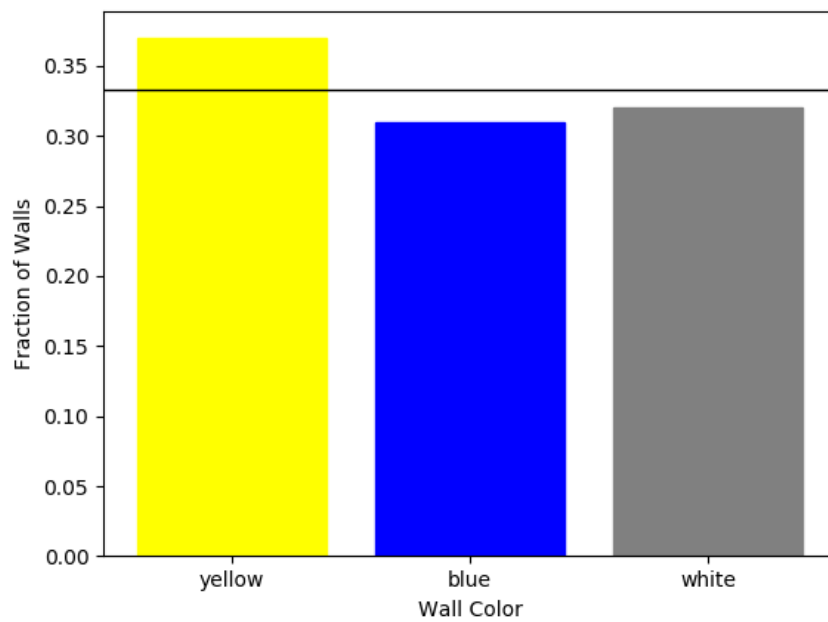


Figure 3.5: Distribution of the color of the floor in the test scenes.

Chapter 4

Proposed Method

We propose a two-part network (which we may refer to as GQN+RL). The first part of the network contains the reinforcement learning algorithm that learns which viewpoints to observe in a given scene, and the second part of the network contains the representation and generation algorithms that produces a representation and renders the predicted image from a randomly selected query viewpoint.

Given a scene, we use a RL agent to propose the next viewpoint and corresponding image that should be passed into the GQN. Posed as a reinforcement learning problem, we let the states $s \in S$ be the raw RGB images to be passed into the GQN and the number of images remaining that can be observed, and the actions $a \in A$ be the viewpoint that the next RGB image is taken from. As depicted in Figure 4.1, in the restricted environment, a is real value α between $[0, 1)$, and in the unrestricted environment, a is 4 real values, each between $[-3.5, 3.5]$, corresponding to the xy location (x_1, y_1) of the agent and the xy location (x_2, y_2) where the agent should be pointing towards. These actions can then be transformed into the appropriate viewpoints v . The reward is 0 until we collect all K observations, in which case the reward is a function of the GQN loss.

From the viewpoint of the reinforcement learning module, the GQN is a black-box algorithm that outputs a reward when given inputs $\{o_i^k\}_{k=1, \dots, K}$. The other module, the GQN, views the agent in a similar fashion: the agent is simply a black box algorithm that generates input images of a scene and the query.

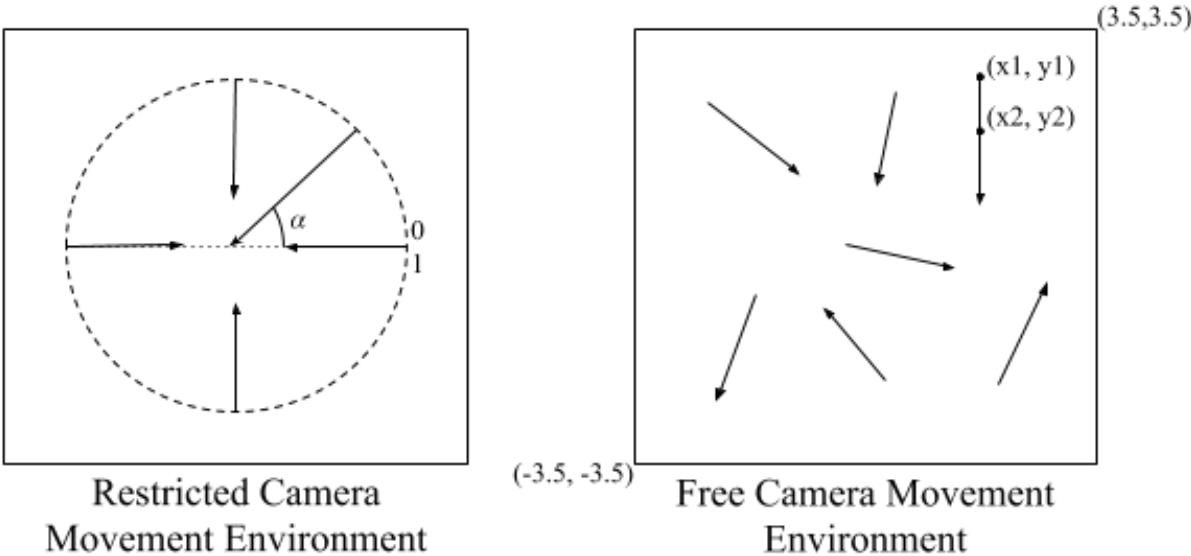


Figure 4.1: The action space of the restricted camera and free camera environments.

4.1 GQN Details

We generally followed the implementation details in [1], with a few modifications to compensate for a significantly lower amount of compute and smaller training datasets. The representation architecture follows the 'tower' convolutional representation architecture (Figure 4.2), as it learns the fastest among all of the variations tested, though many other variations could have been deployed. The generation architecture is modelled after the DRAW network [45] (Figure 4.3), which defines the distribution $g_{\theta}(\mathbf{x}|\mathbf{v}^q, \mathbf{r})$ that the images are sampled from in an iterative fashion. We found that 8 generation iterations to be a good trade-off between speed and accuracy. One note we make is that the choice of generation architecture is extremely flexible; alternatives such as generative adversarial networks (GANs) [31, 32] or auto-regressive models [33] could have been used for the generation architecture. We use the ADAM optimizer [46] with a learning rate of $5 * 10^{-5}$.

4.2 Reinforcement Learning Agent Details

As shown in detail in Figure 4.4, we use a CNN-LSTM network to train our RL agent [34, 35]. We input a 64×64 RGB image x^k into the convolutional neural network and feed its output and the number of observations made so far into a LSTM network. The LSTM outputs the next action, which defines the next viewpoint v^{k+1} . v^{k+1} can then be used to render the corresponding RGB image $x^k + 1$. As mentioned above, the reward is 0 if the number of observations collected so far $k' < K$. If $k' = K$, then the reward is the negative loss of the GQN (negative ELBO). We use PPO [36] to train our agent due to its ease of use and good performance. We set the clipping parameter to 0.2. We use the ADAM optimizer with a learning rate of $5 * 10^{-4}$.

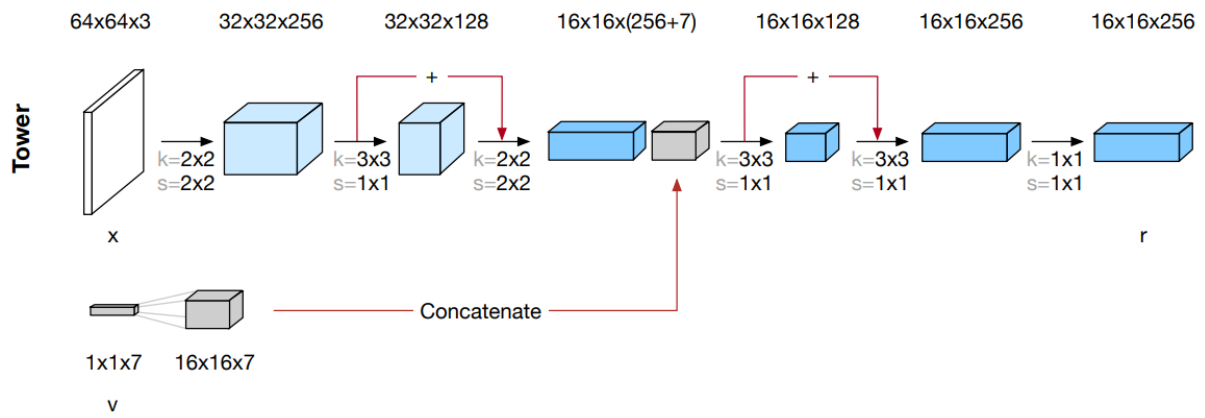


Figure 4.2: The representation network f used for the GQN. This figure is adapted from [1].

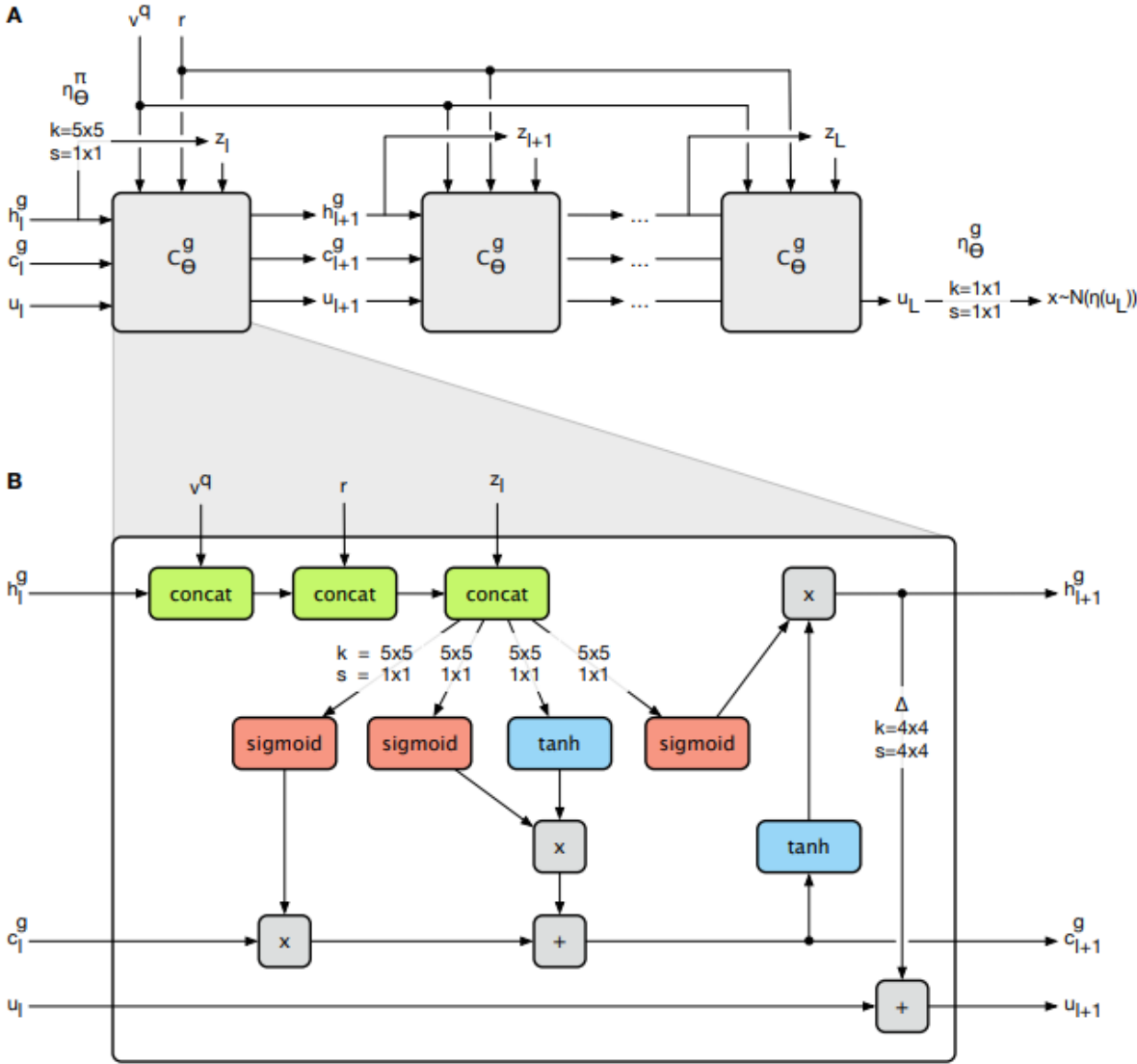


Figure 4.3: The generation network g used for the GQN, which is a modified version of DRAW. This figure is adapted from [1].

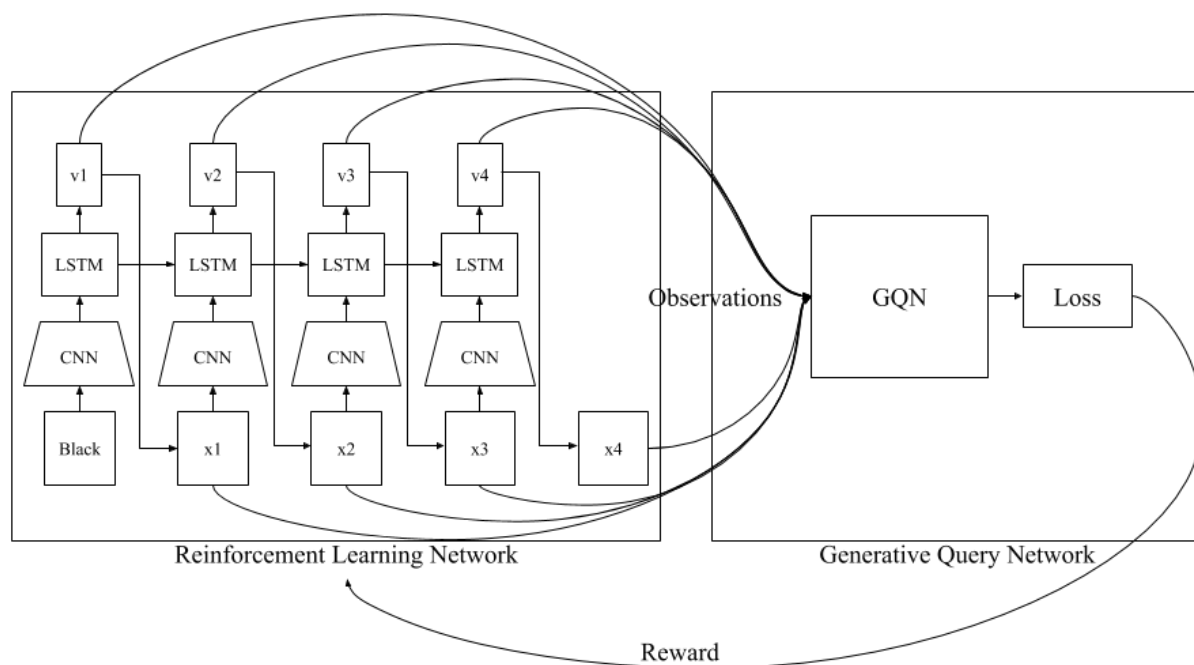


Figure 4.4: The reinforcement learning network and overall pipeline. The CNN takes a $64 \times 64 \times 3$ image as input. The first convolutional layer contains 32 8×8 filters with stride 4, the second layer contains 64 4×4 filters with stride 2, the third layer contains 64 3×3 filters with stride 1, and the last layer is a fully connected layer with 512 units. This is then passed into the LSTM network, which has a hidden state size of 128. We use the ReLU activation function.

Chapter 5

Experiments

5.1 Baseline for Comparison

We can view the original GQN model as an agent that does not learn how to improve its observations, since all viewpoints are chosen randomly. Since we expect the best course of action is to gain as much information from each subsequent observation, we infer that a very good (but not necessarily optimal) strategy is to sample all the viewpoints from the restricted environment uniformly. As visually shown in Figure 5.1, we use the restricted environment to pick viewpoints pointing towards the center of the scene that are evenly spread around the circumference of the circle.

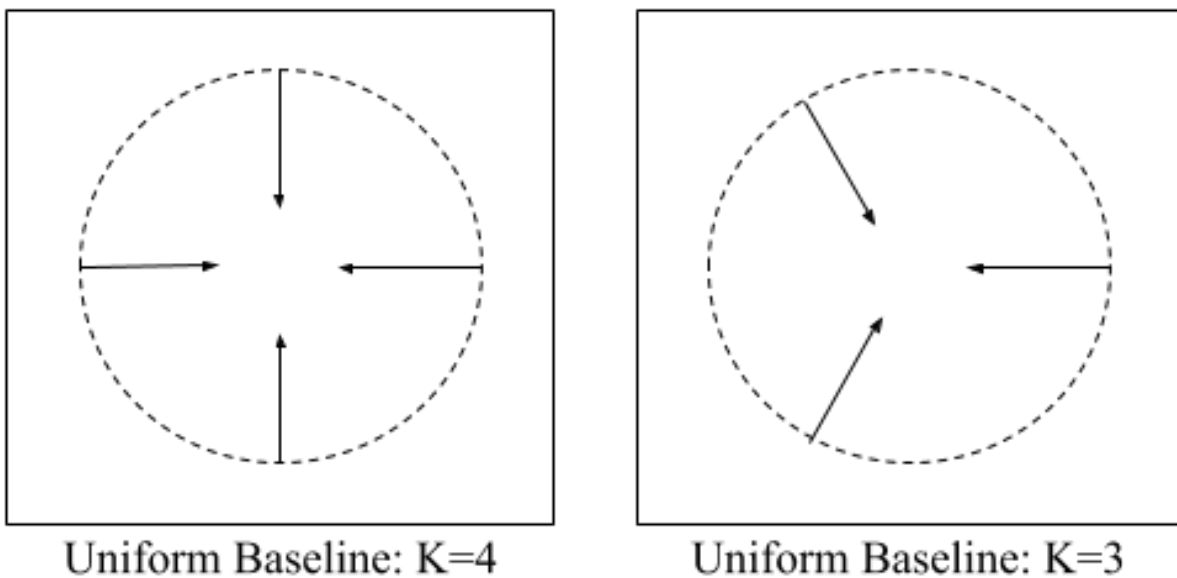


Figure 5.1: Examples of the viewpoints chosen with the baseline.

5.2 Results

After training, we tested each model on the test scenes, and compared the predicted images to the ground truth images corresponding to the query viewpoint, using a per-pixel L2 reconstruction error as the evaluation metric. The training curves of each variation are plotted in Figure 5.2, and the reconstruction results are reported in Table 5.1. Figure 5.3 shows some visual comparisons in example scenes between the GQN and the GQN+RL models in the free environment.

We note that in the restricted environment, the freedom to pick the input observations did not significantly improve the accuracy of the generated images. This is somewhat to be expected, since random observations in the restricted environment are always pointing toward the objects, so there are very few 'bad' or redundant observations. However, in the unrestricted environment, the GQN with random inputs was unable to learn a useful representation, leading to a non-decreasing loss curve and a relatively high reconstruction error. This inability to learn a good representation seems to be caused by the fact that many of the random inputs did not reveal any useful information, likely due to the revelation that many observations were pointed towards the walls of the scene, rather than the objects. However, when we add the ability to pick the inputs, the model is able to generate images on par with those generated in the restricted environment, which suggests that the agent learned to choose observations that pointed towards areas of interest, namely the parts of the room where the objects were.

When only given 2 observations for each scene in the unrestricted environment, we are able to analyze some of the statistical differences between the chosen input viewpoints. Figure 5.4 shows the distribution of the difference of the viewpoint angles between the GQN+RL and the expected distribution of randomly choosing inputs. We see that GQN+RL picks viewpoints that face in opposite directions, as the chosen inputs are usually always on opposite sides of the scene in order to capture the most information in 2 observations. Figures 5.5 and 5.6 show the distances between the viewpoints for GQN and GQN+RL, respectively. GQN+RL effectively avoids capturing observations that are too close together and thus redundant. Instead, it spreads the observations apart when compared to the model with randomly chosen inputs, allowing GQN+RL to learn more about the scene and capture more important features of each room.

We also note that we achieve very similar reconstruction errors to the uniform baseline model, which indicates that our agent selects input observations that generally capture information from a variety of views for each scene. As opposed to the baseline, our model is able to adapt if the objects are not placed near the center of the scene or the scene contains various occlusions, in which case our model would capture more useful observations and learn a better scene representation.

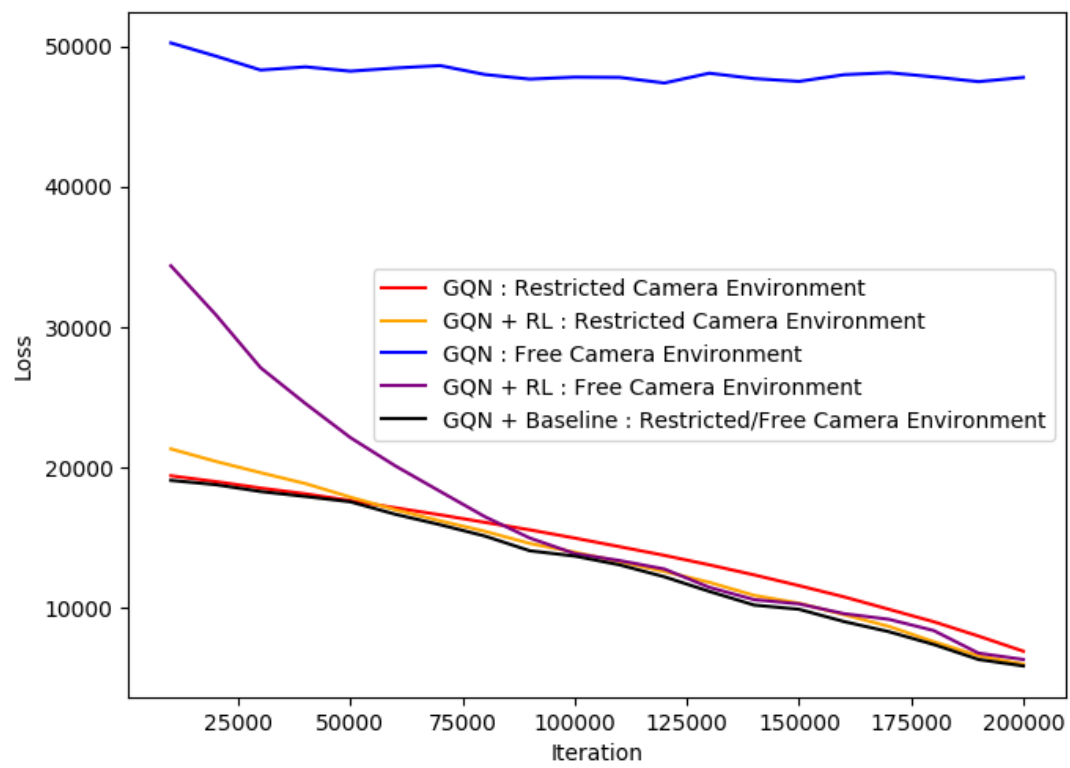


Figure 5.2: Training curves.

Table 5.1: Comparison of average reconstruction errors.

Experiment		
Model	Environment	L2 Reconstruction Error ($\times 10^{-3}$)
GQN + Baseline	Restricted/Free	1.797
GQN	Restricted	2.073
GQN + RL	Restricted	1.843
GQN	Free	41.906
GQN + RL	Free	1.988

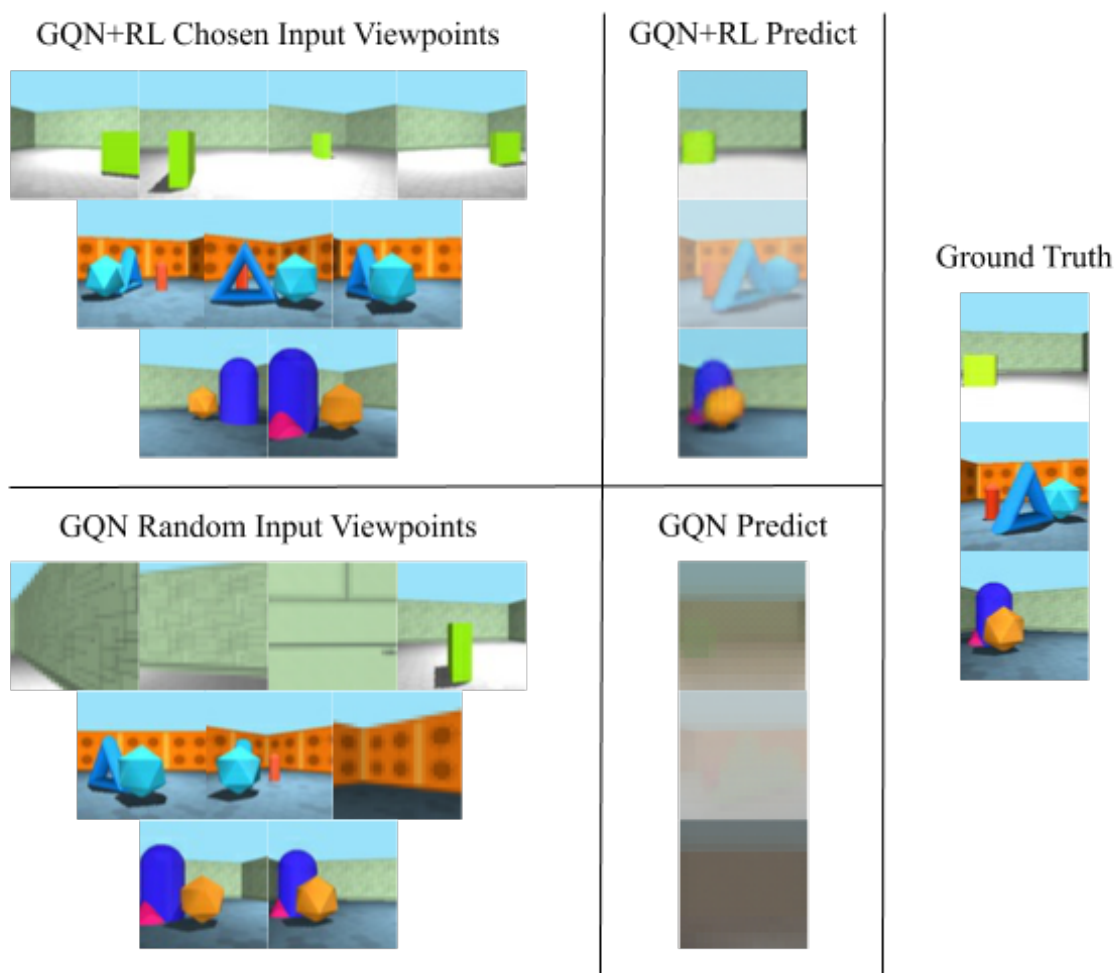


Figure 5.3: Comparison of GQN versus GQN+RL input viewpoints and image reconstructions in the free environment. Ground truth refers to the rendered image at the query viewpoint. GQN+RL and GQN predict are the respective models' generated images at the query viewpoint.

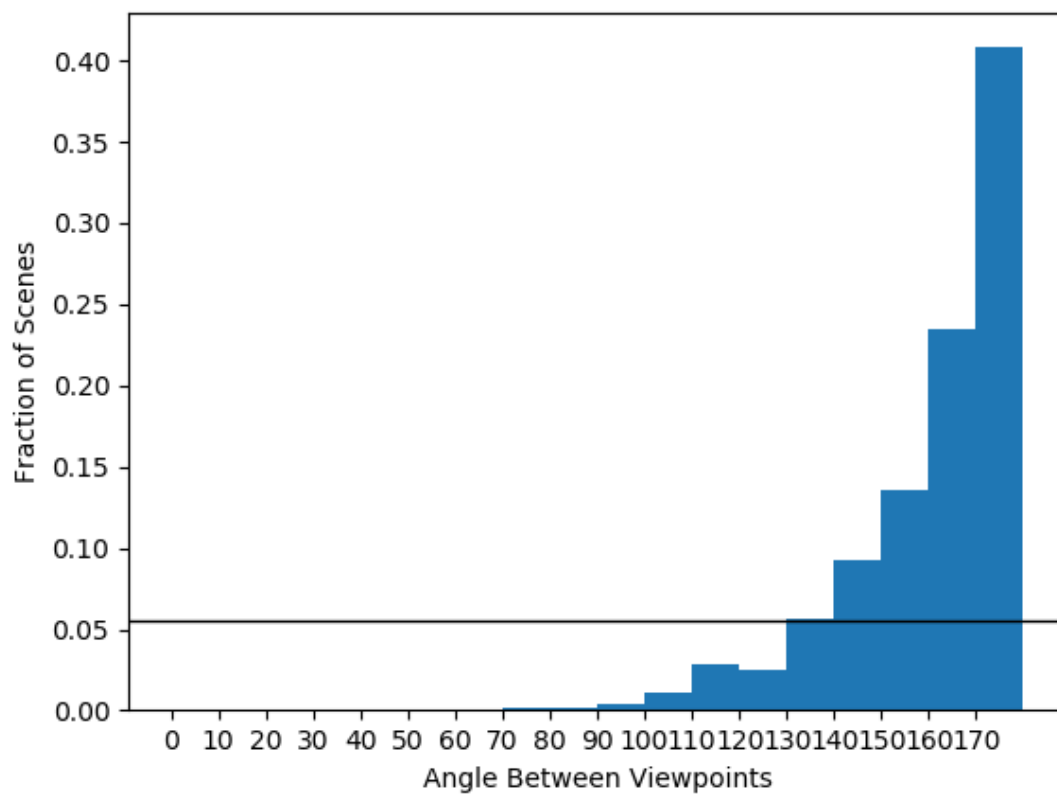


Figure 5.4: The distribution of the difference between angles of the RL chosen viewpoints when only 2 observations are allowed (10 degree bins) in the unrestricted environment. The black line indicates the distribution of random observations.

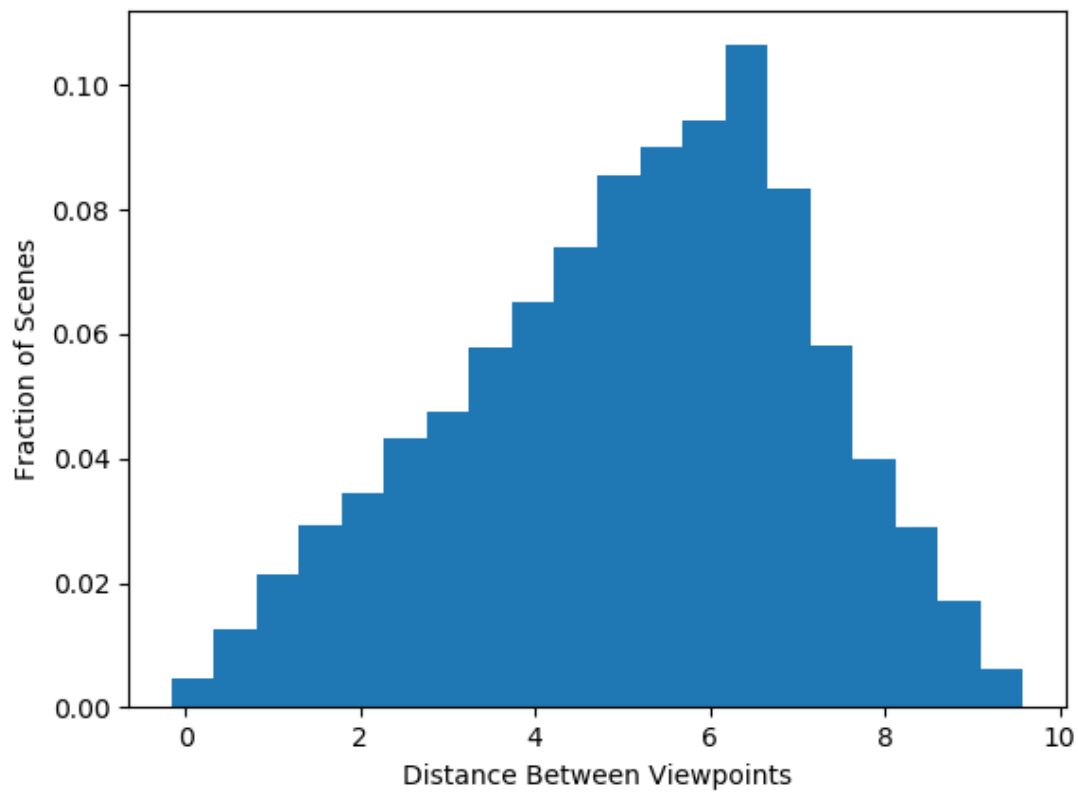


Figure 5.5: The distribution of the distance between the randomly chosen viewpoints when only 2 observations are allowed in the unrestricted environment.

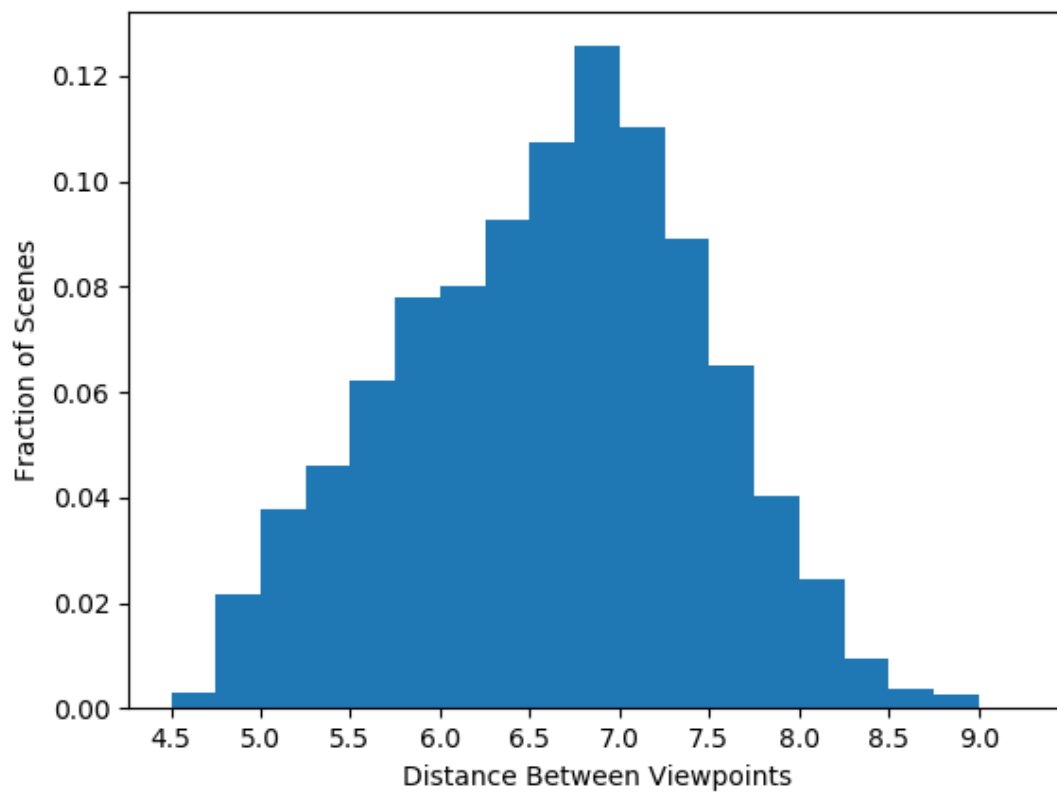


Figure 5.6: The distribution of the distance between the RL chosen viewpoints when only 2 observations are allowed in the unrestricted environment.

Chapter 6

Conclusion and Future Work

In this work, we have extended the GQN framework to allow an agent to pick the input observations rather than rely on random observations. Without losing any of the attractive properties that GQN offers, such as view-independent representations or flexibility in generation architectures, we are able to improve the reconstruction of complex scenes by training an agent to navigate through the environment and select observations that offer more information not captured by the previous observations. The vast improvement to our restriction-free environment suggests that our extension can be employed to improve the representations learned in highly complex environments, such as GTA V and Minecraft; learning representations in dynamic environments that change over time could also be explored.

While we have shown promising results with our extended framework, we can further improve the model by using a reinforcement learning algorithm that is more parallelizable, so that we can take advantage of the exponential increases in compute power. More complex networks can also be substituted in and experimented with in order to achieve better representations. In addition, we can also work on removing a simplifying assumption that the agent can sample any viewpoint that it wants, and penalize observation viewpoints that take a long time for the agent to reach. We can also build or adapt more complex environments such as self-driving simulators in order to test the GQN+RL model’s ability to learn good scene representations in very dense and variable environments.

We believe that many problems in complex environments can only be effectively solved when our models are able to identify the most important features of the environment, in which scene representation and our extension to the GQN framework is a major step towards this goal.

Chapter 7

References

- [1] S. M. A. Eslami, et al. Neural Scene Representation and Rendering. In *Science*, 2018.
- [2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator, In *CoRL*, 2017.
- [3] M. Martinez, C. Sitawarin, K. Finch, L. Meincke, A. Yablonski, and A. L. Kornhauser. Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars. *preprint arXiv:1712.01397*, 2017.
- [4] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. Scenenet: Understanding real world indoor scenes with synthetic data, in *CVPR*, 2016.
- [5] S. Song, S. Lichtenberg, and J. Xiao. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In *CVPR*, 2015.
- [6] A. Chang, et al. ShapeNet: An Information-Rich 3D Model Repository. *preprint arXiv:1512.03012*, 2015.
- [7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.
- [8] R. Girshick. Fast R-CNN. In *ICCV*, 2015.
- [9] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015.
- [10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection. In *CVPR*, 2016.
- [11] L. He, G. Wang, and Z. Hu. Learning Depth from Single Images with Deep Neural Network Embedding Focal Length. *preprint arXiv:1803.10039*, 2018.
- [12] J. Bontar, and Y. LeCun. Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. In *JMLR*, 2016.
- [13] S. Song, et al. Semantic Scene Completion from a Single Depth Image. *preprint arXiv:1611.08974*, 2016.
- [14] J. Gurin, et al. Semantically Meaningful View Selection. In *IROS*, 2018.

- [15] Z. Wu, et al. 3D ShapeNets: a deep representation for volumetric shapes. In *CVPR*, 2015.
- [16] J. Wu, C. Zhang, T. Xue, W. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, 2016.
- [17] D. J. Rezende, et al. Unsupervised learning of 3D structure from images. In *NIPS*, 2016.
- [18] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: learning single-view 3D object reconstruction without 3D supervision. In *NIPS*, 2016.
- [19] M. Pollefeys, et al. Visual modeling with a hand-held camera. In *IJCV*, 2004.
- [20] Y. Zhang, W. Xu, Y. Tong, and K. Zhou. Online structure analysis for real-time indoor scene reconstruction. In *ACM Transactions on Graphics*, 2015.
- [21] D. P. Kingma, and M. Welling. Auto-Encoding variational Bayes. In *ICLR*, 2013.
- [22] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic back-propagation and variational inference in deep latent Gaussian models. In *ICML*, 2014.
- [23] I. Goodfellow, et al. Generative adversarial nets. In *NIPS*, 2014.
- [24] K. Gregor, F. Besse, D. J. Rezende, I. Danihelka, and D. Wierstra. Towards conceptual compression. In *NIPS*, 2016.
- [25] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.
- [26] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The Helmholtz machine. In *Neu. comp.*, 1995.
- [27] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to predict new views from the worlds imagery. In *CVPR*, 2016.
- [28] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.
- [29] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: a physics engine for model-based control. In *IROS*, 2012.
- [30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *preprint arXiv:1606.01540*, 2016.
- [31] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *preprint arXiv:1710.10196*, 2017.
- [32] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *preprint arXiv:1701.07875*, 2017.
- [33] A. v. d. Oord, et al. Conditional image generation with PixelCNN Decoders. In *NIPS*, 2016.
- [34] S. Hochreiter, and J. Schmidhuber. Long short-term memory. In *Neural Comput.*, 1997.
- [35] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *IPS*, 2012.
- [36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *preprint arXiv:1707.06347*, 2017.

- [37] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. *preprint arXiv:1801.08757*, 2018.
- [38] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *NIPS*, 2016.
- [39] A. Tampuu, et al. Multiagent Cooperation and Competition with Deep Reinforcement Learning. In *PLoS ONE*, 2017.
- [40] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative Multi-Agent Control Using Deep Reinforcement Learning. In *AAMAS*, 2017.
- [41] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS*, 2017.
- [42] K. He, G. Gkioxari, P. Dollr, and R. Girshick. Mask R-CNN. *preprint arXiv:1703.06870*, 2017.
- [43] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In *RSS*, 2018.
- [44] R. Bellman. The theory of dynamic programming. In *Bulletin of the American Mathematical Society*, 1954.
- [45] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. DRAW: A Recurrent Neural Network For Image Generation. In *PMLR*, 2015.
- [46] D. P. Kingma, and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.
- [47] Q. Zhang, and S. Zhu. Visual Interpretability for Deep Learning: A Survey. *preprint arXiv:1802.00614*, 2018.
- [48] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. In *IEEE Transactions on Robotics*, 2016.
- [49] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *IROS*, 2015.
- [50] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *preprint arXiv:1611.01224*, 2016.
- [51] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *preprint arXiv:1509.02971*, 2015.