SYNOPSIS OF

Approaches to Spatial Reasoning in Reinforcement Learning

A Project Report

submitted by

SAI PRAVEEN B

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 1

Contents

1 Introduction

1.1 Deep Generative Models

1.2 Deep Projection Networks

2 Previous Work

3 Motivation

3.1 Video Prediction

3.2 Convolutional Networks

3.3 Ray Tracing-inspired Network

4 Background

4.1 Reinforcement Learning

4.2 Generative Models

4.3 Gaussian-Binary Restricted Boltzmann Machines

RBMs have been used widely to learn energy models over an input distribution $p(\mathbf{X})$. RBM is an undirected, complete bipartite, probabilistic graphical model with N_h hidden units, H, and N_v visible units, V. In Gaussian-Binary RBMs, hidden units are binary units(Bernoulli distribution) capable of representing a total of 2^{N_h} combinations, while the visible units use the Gaussian distribution. The network is parametrized by edge weights matrix $\mathbf{W}$ between each node of V and H, and bias vectors $\mathbf{a}$ and $\mathbf{b}$ for V and H respectively. Given a visible state $\mathbf{v}$,

the hidden state, $\mathbf{h}$, is obtained by sampling the posterior given by

$$p(\mathbf{h}|\mathbf{v}) = \frac{1}{1 + \exp(\mathbf{W}^T\mathbf{v} + \mathbf{b})}$$

Given a hidden state $\mathbf{h}$, visible state $\mathbf{v}$ is obtained by sampling the posterior given by

$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{W}^T\mathbf{h} + \mathbf{a}, \mathbf{\Sigma})$$

Since RBMs model conditional distributions, conditional distributions($p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$) have a closed form while marginal and joint distributions($p(\mathbf{v})$, $p(\mathbf{h})$ and $p(\mathbf{h}, \mathbf{v})$) are impossible to compute without explicit summation over all combinations.

Parameters are learnt using contrastive divergence((**?**)). Learning $\mathbf{\Sigma}$, however, proved to be unstable((**?**)) and hence, we treat σ as a hyperparameter and use $\mathbf{\Sigma} = \sigma * \mathcal{I}_{N_v}$.

4.4 Variational Auto-encoders

Variational Auto Encoders(VAE)(**?**) attempt to learn the distribution that generated the data $\mathbf{X}$, $p(\mathbf{X})$. VAEs, like standard autoencoders have an encoder, $\mathbf{z} = f_e(\mathbf{x})$, and a decoder $\mathbf{y} = f_d(\mathbf{z})$ component. Generative models that attempt to estimate $p(\mathbf{X})$ use a likelihood objective function, $p_\theta(\mathbf{X})$ or $\log p_\theta(\mathbf{X})$. More formally, the objective function can be written as

$$p(\mathbf{x}) = \int_{\mathbf{Z}} p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z}) \approx \sum_{z \sim \mathbf{Z}} p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z})$$
$$\hat{p}(\mathbf{x}) \approx \sum_{z \in \mathbf{D}} p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z}) \text{ where } D = \{\mathbf{z_1}, \mathbf{z_2} \ldots, \mathbf{z_m}\} \text{ and } \mathbf{z}_k \sim \mathbf{Z} \, \forall k$$

where $p(\mathbf{x}|\mathbf{z};\theta)$ is defined to be $\mathcal{N}(f(\mathbf{z};\theta), \sigma^2\mathcal{I})$.
Gradient-motivated learning requires approximation of the integral with samples. In high-dimensional $\mathbf{z}$-space, this could lead to large estimation errors as $p(\mathbf{x}|\mathbf{z})$ is likely to be concentrated around a few select $\mathbf{z}$s and it would take an infeasible number of samples to get proper estimate. VAEs circumvent this problem by introducing a new distribution $\mathcal{Z} \sim \mathcal{N}(\mu_\phi(\mathbf{z}), \sigma_\phi(\mathbf{z}))$ to sample D from. To reduce parameters, we use $\sigma_\phi(\mathbf{z}) = c \cdot I$. These two functions are approximated with a deep network and form

the *encoder* component of the VAE. $p_\theta(\mathbf{X})$ is represented using the sampling function $f(z;\theta)$ where $z \sim \mathcal{N}(0,1)$ and $f(z)$ forms the *decoder* component of the VAE. After some mathematical sleight of hand to account for $\mathcal{Z}$ in the learning equations((**?**) provides an intuitive understanding of these equations), we obtain the following formulation of the loss function

$$E = \mathcal{D}_{\mathcal{KL}}(\mathcal{N}(\mu(\mathbf{X}), \sigma(\mathbf{X})), \mathcal{N}(0,1)) + \sum_{k=1}^{N} \| \frac{(\mathbf{y}_i - \mathbf{x}_i)}{\sigma}^2 \|$$

where the KL-divergence term exists to adjust for importance sampling $\mathbf{z}$ from $\mathcal{Z}$ instead of $\mathcal{N}(0,1)$.

4.5 3D Scene Generation

Classical Computer Graphics typically deals with two broad areas: Projection and Pixel-Shading.

The former, *projection* deals with transforming the visible 3D scene into the viewport of the camera (aka, a 2D plane/representation). In an abstract sense, projection simply reduces the dimensionality by performing an irreversible transformation to a simpler space.

The latter, *pixel-shading* deals with calculating the color of each pixel in the final scene. For the purposes of this thesis, we ignore this part (all surfaces in our world have a single, solid color) as it's irrelevant to our objective.

Projection, in practice, is done in two different modes

1. Orthographic Projection: Orthographic projection (or parallel projection) involves directly projecting every vertex along the perpendicular onto the target plane. This projection maintains the relative distances between points, but for practical purposes produces an extremely unnatural image of the scene. the fact that perspective distance to the plane does not affect the projection of a vertex makes it impossible to pick up depth cues through parallax.
   These reasons, combined with the fact that both video games and real-life cameras are represented by the perspective projection, justify why this thesis focuses mostly on the perspective projection.
   However, the demonstrated architecture can just as easily be used for any projection with certain properties (which will be stated soon).
2. Perspective Projection:

Figure 1: Orthographic Projection

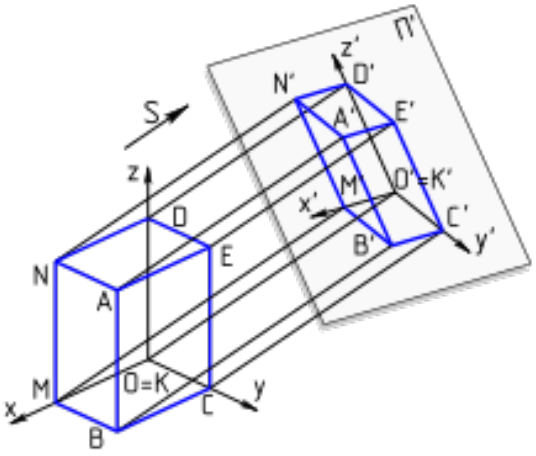
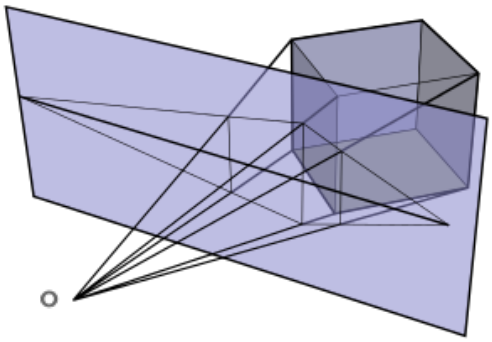


Figure 2: (Weak) Perspective Projection



A simple form of the perspective projection simply transforms the 3D point into the tangent of the angle made by the point to the plane along X Y axes.
The following diagram presents a simple definition of Weak Perspective Projection:

Figure 3: (Weak) Perspective Projection

TODO: change the line and point labels.
For a fixed vanishing point, C, the weak perspective projection of any other point D is simply the intersection of the line DC with the plane perpendicular to the camera direction and at a distance of *unity* from the camera.

The following equations outline the transformation from camera-space 3D coordinates (X_c, Y_c, Z_c) to pixel space (with the vanishing point as the origin):

$$P_x = \frac{\frac{X_a}{Z_a} + 1}{2}.W$$

$$P_y = \frac{-\frac{Y_a}{Z_a} + 1}{2}.H$$

If the origin $(0, 0, 0)$ and the Z-axis are not the camera position and direction respectively, an additional linear transformation is required to get the world-space coordinates into camera-space (commonly referred to as the view matrix)

$$(X_c, Y_c, Z_c) = M_v.(X_w, Y_w, Z_w)$$

A third, more important projection is the (Strong) Perspective projection. The main difference is that the strong perspective transform does not consider points that are either too near ($z < z_{near}$) or too far ($z > z_{far}$). Although we find the strong perspective projection widely used in games, for simplicity, this thesis limits itself to the weak perspective projection.

4.6 Ray Tracing

Ray tracing (or more simply, ray casting) is a technique where a ray corresponding to each of the pixels of the final image is traced through a 3D scene and is tested for intersection with an object.

Every pixel in an image corresponds to a ray in 3D space which depends on the projection mechanism used for this particular system.

The algorithm to form the ray uses the inverse of the perspective projection algorithm.

The general one-step ray tracing algorithm works as follows:

Algorithm 1 Pseudo-code that demonstrates the process of projecting a 3D scene onto a 2D representation

```
1:  procedure RENDERPIXEL(p_x, p_y)
2:      ray ← GetRay(pixel_x, pixel_y)
3:      curr_depth ← ∞
4:      curr_intersection ← null
5:      for do obj in objects
6:          intersection ← RayObjectIntersect(ray, object)
7:          if intersection.depth < curr_depth then
8:              curr_depth = intersection.depth
9:              curr_intersecion = intersection
10:         if curr_intersection ≠ null then
11:             return curr_intersection.color
```

Algorithm 2 Pseudocode that demonstrates the process of projecting a 3D scene onto a 2D representation

1: **procedure** GETRAY(p_x, p_y)
2: $h_x \leftarrow 2.\frac{p_x}{W} - 1$
3: $h_y \leftarrow -2.\frac{p_y}{H} + 1$

4.7 Deep Q-learning

5 Fully Observable Spatial Reasoning

5.1 RBM Model

(fill it in) (CS6700 project)

8 Publications

8.1 Papers in Workshops

1. Title
   Author 1