Project Report #5

Smart Car with World Models

David Ha and Jurgen Schmidhuber
(Google Brain)
Code Tutorial with David Foster

劉美忻 (105061807)
June 3, 2018

GOALS

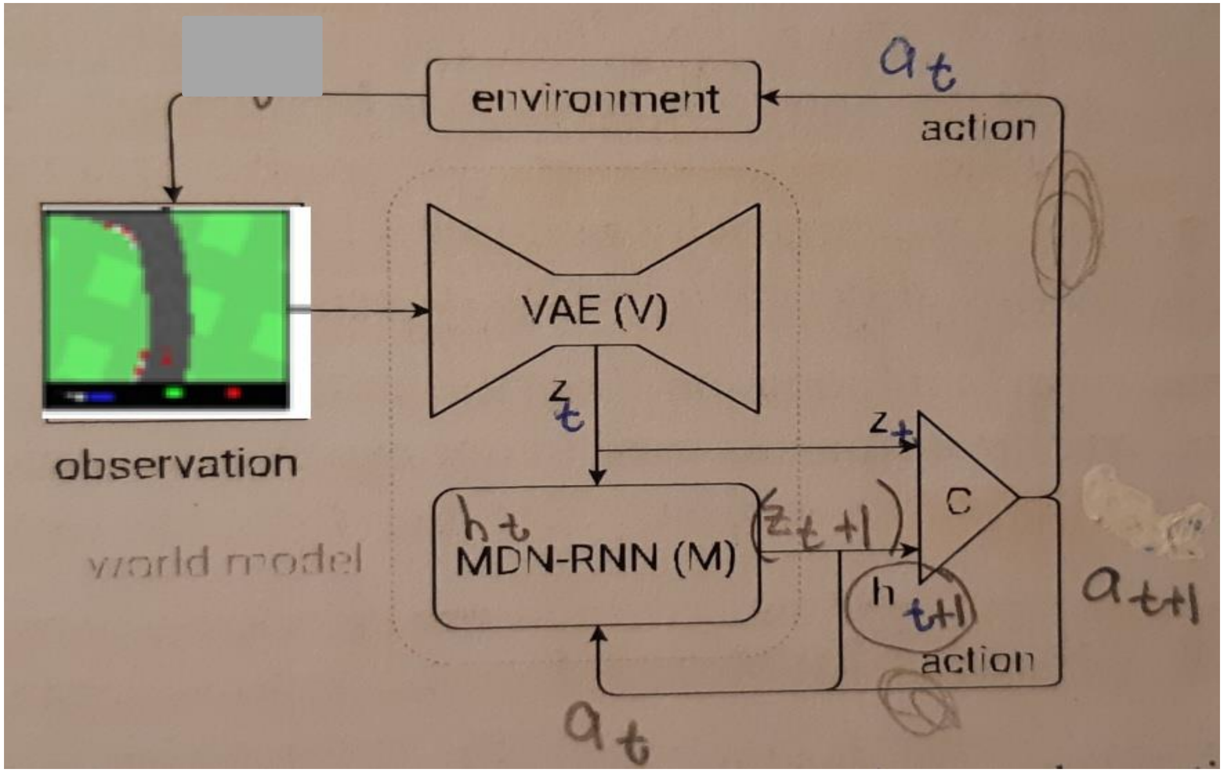
- **A**lgorithms:
  - (VAE) Variational Autoencoder
  - (MDN-RNN) Mixture Density Network Recurrent Neural Network
  - (CMA-ES) Covariance Matrix Adaptation Evolution Strategy.
- **B**ig Data:
  - Open AI Gym car-racing environment. Generate 600,000 images of 64x64 RGB track views, with car action, and what the next frame is.
  - Examples of the VAE compressed and reconstructed images used for training
- **C**ode:
  - Keras framework with Tensorflow, open source code accompanying the paper
  - Installation details and problems encountered with Linux commands and running code on remote server (GoogleCloud). Hardware limitations for 'free trial' and time considerations.
  - Results from two training attempts
  - Ideas for further exploration

World Models

- Interactive paper https://worldmodels.github.io/
- printable PDF of the paper https://arxiv.org/abs/1803.10122
- code repository https://github.com/AppliedDataSciencePartners/WorldModels

- Code Tutorial by David Foster on Medium Daily Digest: *Hallucinogenic Deep Reinforcement Learning Using Python and Keras*
  https://medium.com/applied-data-science/how-to-build-your-own-world-model-using-python-and-keras-64fb388ba459

World Models Paper



- Goal: Drive the car around the track accurately and fast.
- Reward: gain points for gray tiles visited, lose points for timesteps. >900 out of 1000 is considered passing.
- Based on pixel input, decide on the action: steer, accelerate, brake.

World Models Paper

1. Collect 10,000 rollouts from a random policy.
2. Train VAE (V) to encode frames into $z \in \mathcal{R}^{32}$.
3. Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$.
4. Define Controller (C) as $a_t = W_c\ [z_t\ h_t]\ +\ b_c$.
5. Use CMA-ES to solve for a W_c and b_c that maximizes the expected cumulative reward.

Model	Parameter Count
VAE	4,348,547
MDN-RNN	422,368
Controller	867

- Complexity is in the World Model (V and M) ~ expressiveness
  - Backpropagation and gradient descent
- The controller (C) is has fewer parameters so we can explore with less traditional Evolution Strategy to replace the more traditional Reinforcement Learning methods.

VAE (Variational AutoEncoder)

- 64x64 RGB pixel image → 32-dimensional ‘z’
- Compressed, faster
- Feature engineering
  - Speech MFCC?
  - Face features?

A.1. Variational Autoencoder

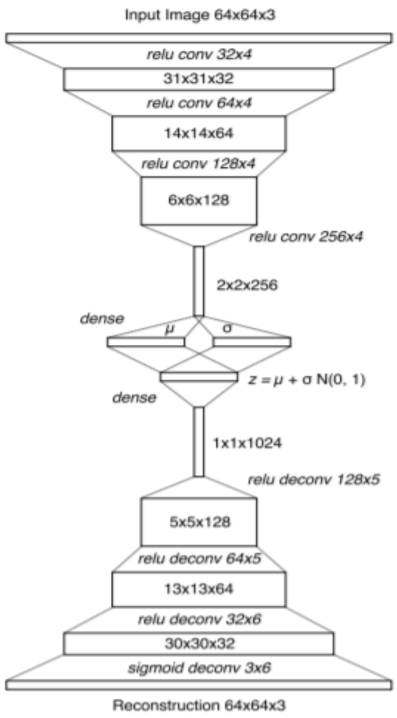
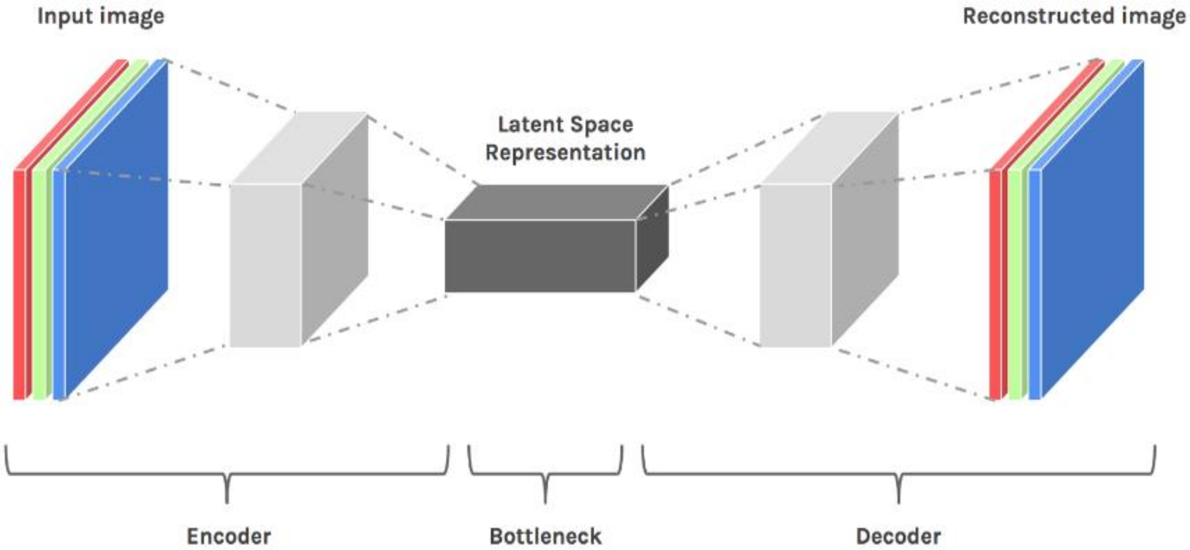


Figure 22. Description of tensor shapes at each layer of ConvVAE.

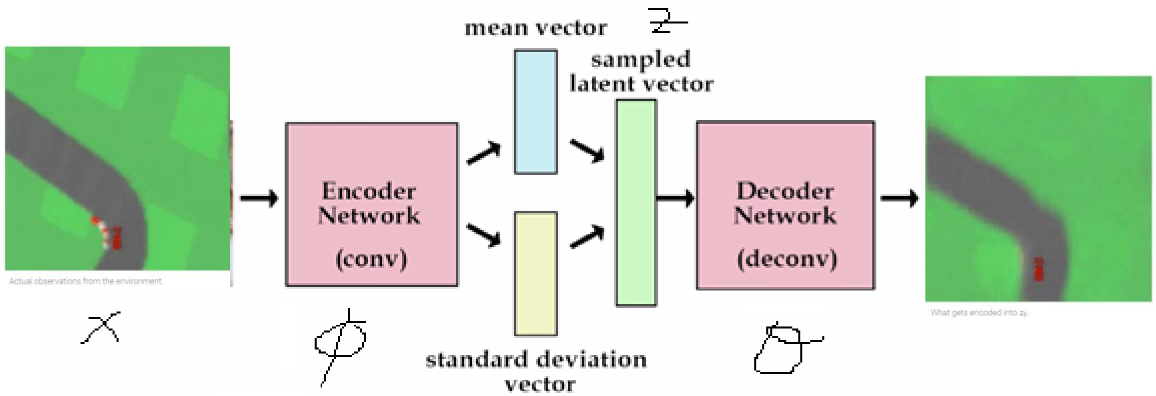
VAE

- Auto-encoder



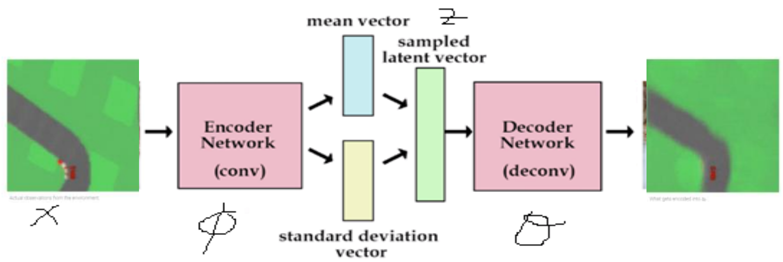
Convolutional Encoder-Decoder architecture

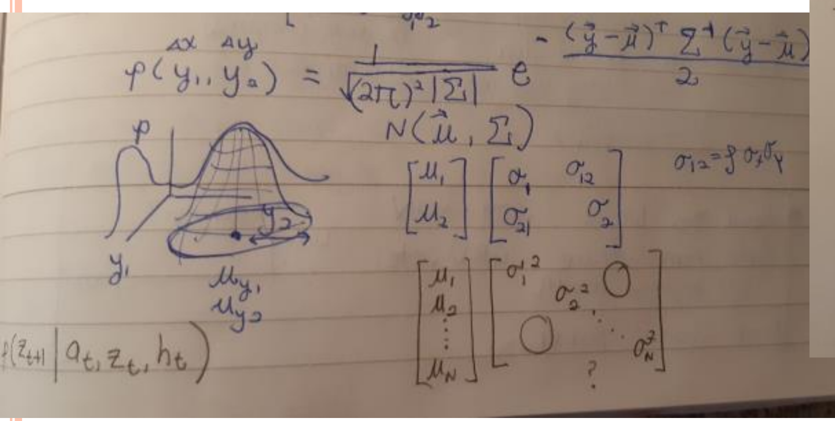
- Variational
  - the bottleneck represents a distribution from which the compressed vector is sampled.

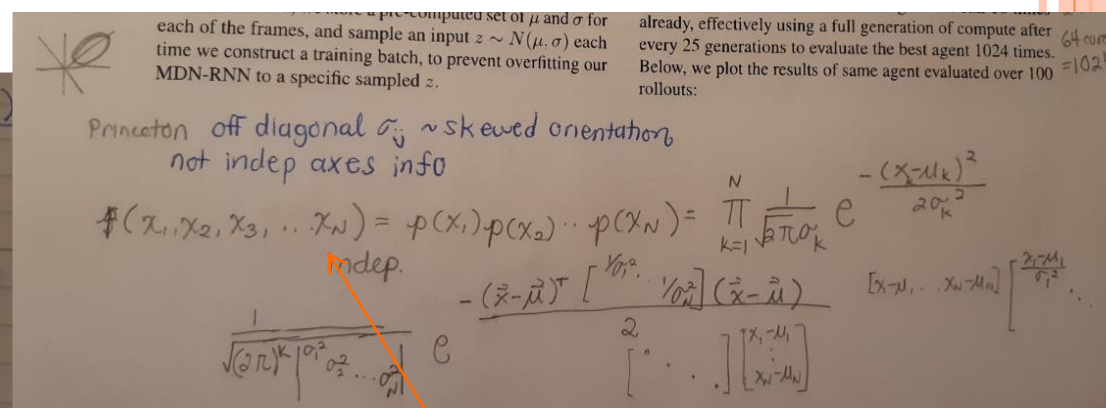


VAE

- The sampling is from a single diagonal Gaussian distribution.
- Enforcing a Gaussian prior makes the world model more robust to unrealistic z vectors

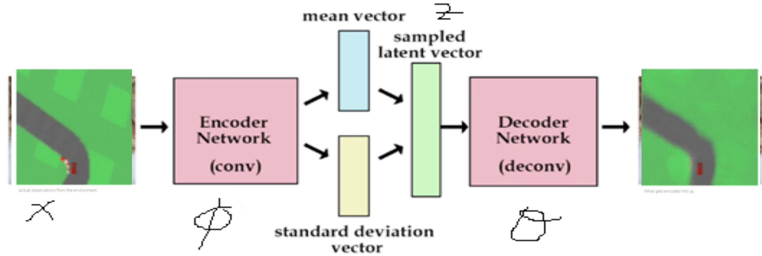




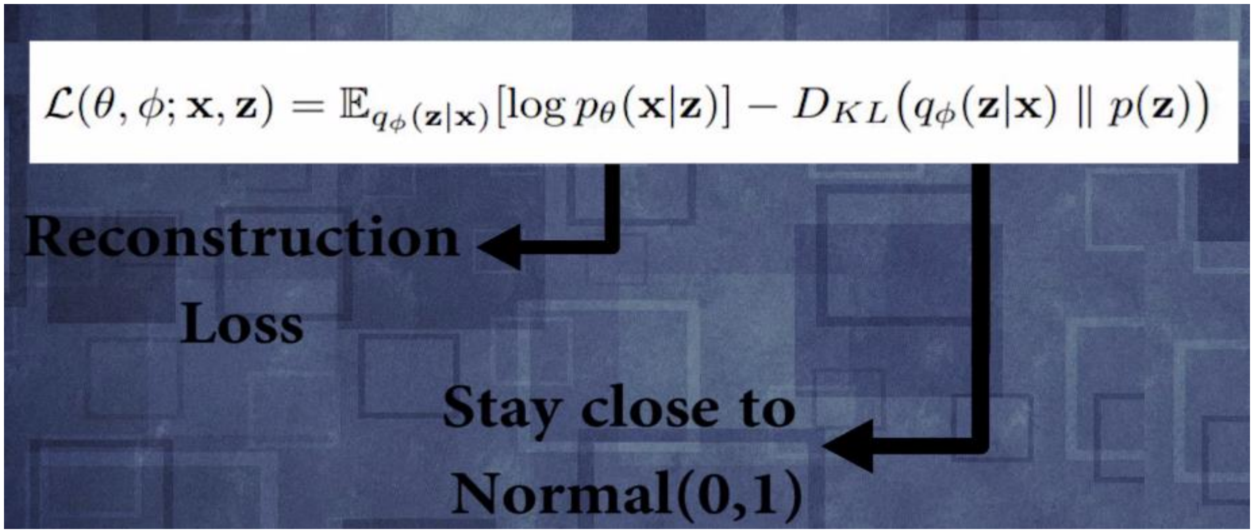


- actually z here

VAE

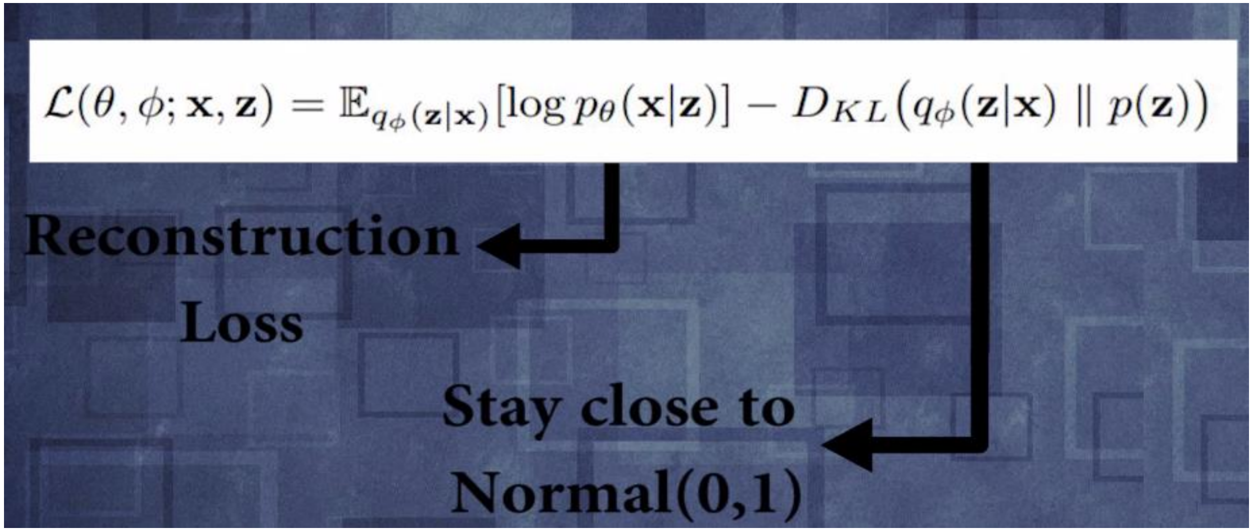


- The goal is to maximize the likelihood L given by:



- First term: expected log likelihood that the decoder outputs the x of the original using the trained theta and the bottleneck sampled z, but z itself is stochastic with probability from the gaussian q~ N(mu, sigma) which depends on how phi processes the input x.
- The second term is the Kullback-Liebler divergence between 2 distributions. A low number means that we keep the distribution *q* from which z is sampled to be close to N(0,1)

VAE



- This project's code replaces maximizing likelihood with minimizing loss function
- **`loss = vae_r_loss + vae_kl_loss`**
- **`vae_r_loss`** = mean square error (L2 distance between input image and reconstructed image)
- **`vae kl loss`** = log of the KL divergence

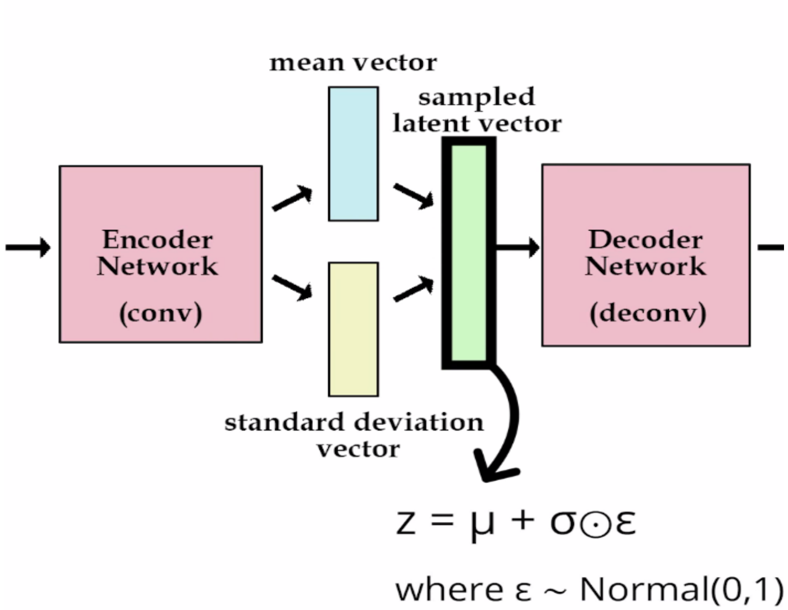
$$D_{\mathrm{KL}}(\mathcal{N}_0 \| \mathcal{N}_1) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^{\mathrm{T}}\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \ln\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\right)$$

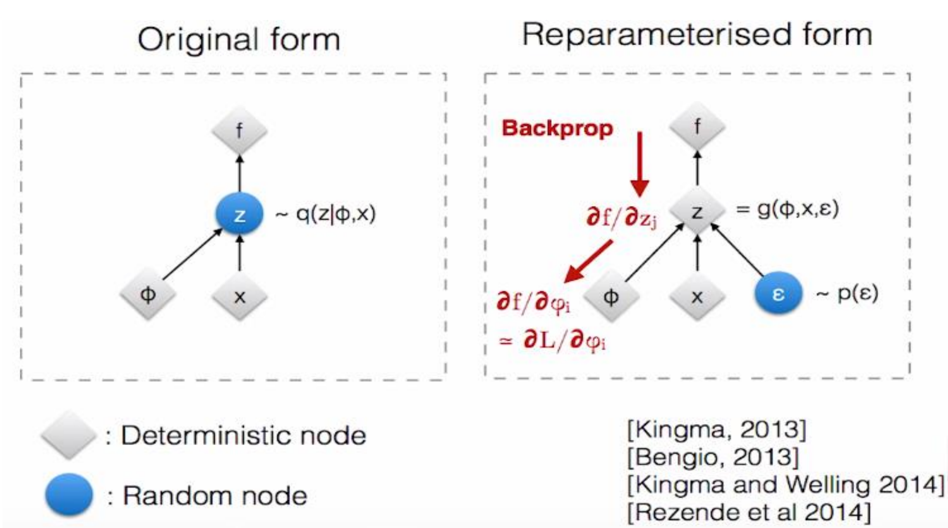
https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#Multivariate_normal_distributions

VAE

- Backpropagation calculate the gradient for descent, with RmsProp algorithm (fancy gradient descent with normalization by root mean square of a moving average of gradients)
- How is the loss function differentiable if there is sampling of the z-vector?
- Reparameterization trick

VAE: Reparameterization Trick





- The mean and sigma are learned parameters to train, but the stochastic part is put in the epsilon, which is a fixed stochastic node that does not need backpropagation to run through.
- Thus, instead of a fully stochastic node in the way that blocks the back propagation, the reparameterized form allows the gradients to get back to the parameters we are interested in training.

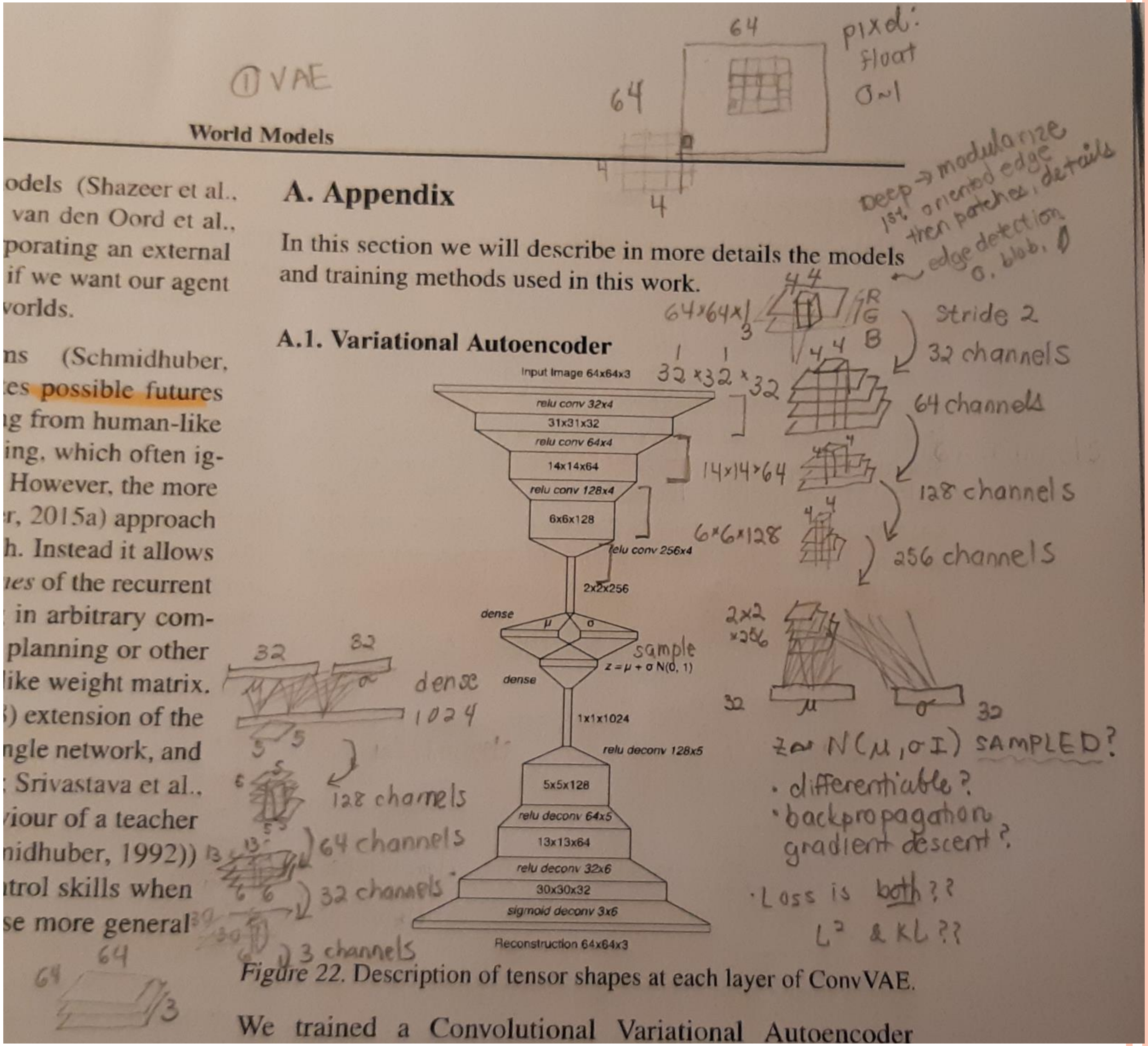
VAE

- 64x64 RGB pixels

compressed

into 32 dims

z that follows a Gaussian distribution

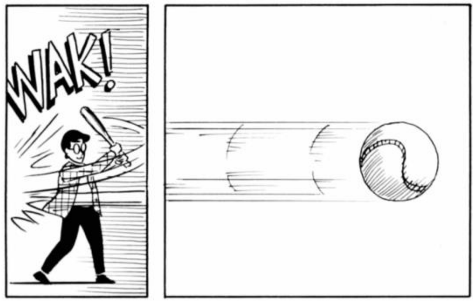


VAE

- Further Reading
  - https://www.youtube.com/watch?v=9zKuYvjFFS8 (15 minute introduction to Variational Autoencoders)
  - Kingma and Welling's May 2014 paper *Auto-Encoding Variational Bayes* https://arxiv.org/abs/1312.6114
  - KL divergence of gaussians
    http://www.allisons.org/ll/MML/KL/Normal/
    https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#Multivariate_normal_distributions
  - Deconvolution
    https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers

MDN-RNN

- RNN (Recursive Neural Network)
  - Sequence
  - Without the prediction of what to expect in next frame after an action, we have erratic wobbly driving
  - 256 hidden neurons
  - carracing_z_only.mp4



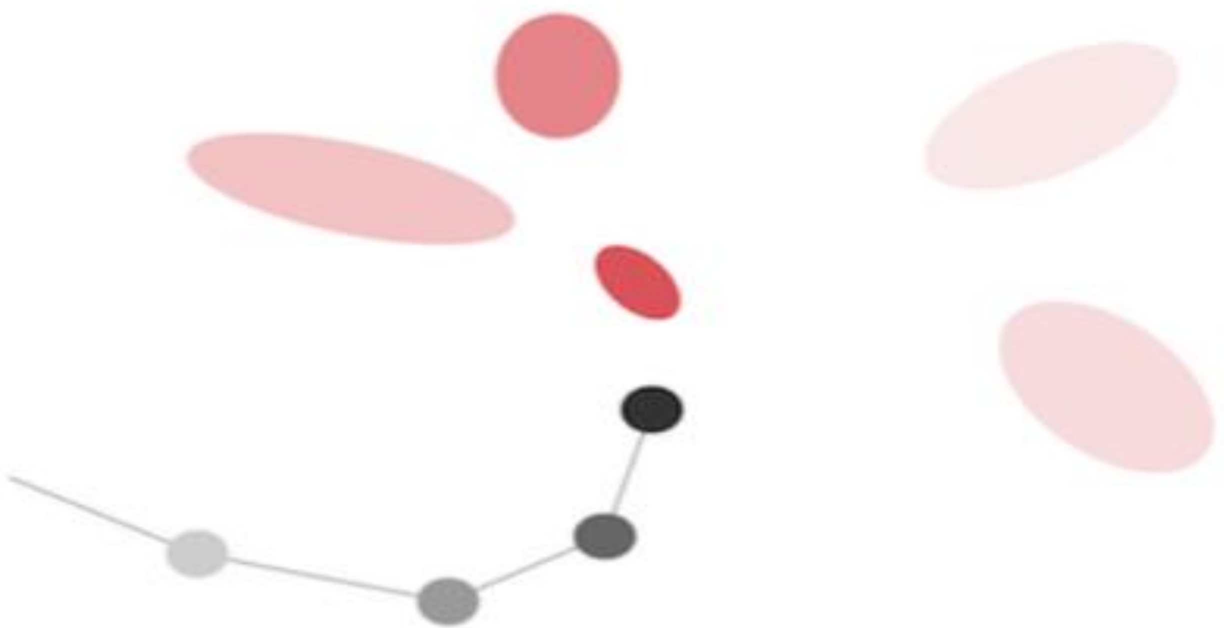
METHOD	AVG. SCORE
DQN (PRIEUR, 2017)	343 ± 18
A3C (CONTINUOUS) (JANG ET AL., 2017)	591 ± 45
A3C (DISCRETE) (KHAN & ELIBOL, 2016)	652 ± 10
CEOBILLIONAIRE (GYM LEADERBOARD)	838 ± 11
V MODEL	632 ± 251
V MODEL WITH HIDDEN LAYER	788 ± 141
FULL WORLD MODEL	**906 ± 21**

Table 1. `CarRacing-v0` scores achieved using various methods.

  - No memory (input z only, no h) vs. with memory (z and h) input to controller

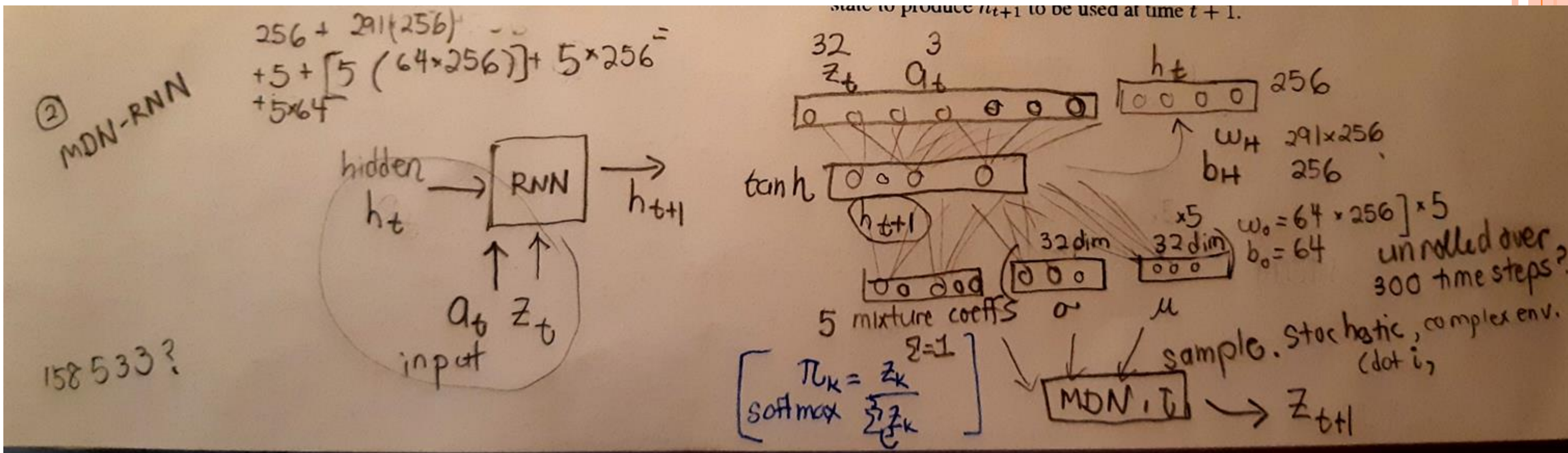
MDN-RNN

- MDN (Mixture-Density Network)
  - Not just predicting the next frame, we allow the next frame's 'image' to be from one of 5 gaussian distributions.
  - Dotting the i in handwriting generation.
  - Doom game, switching mode to fireball start-up.



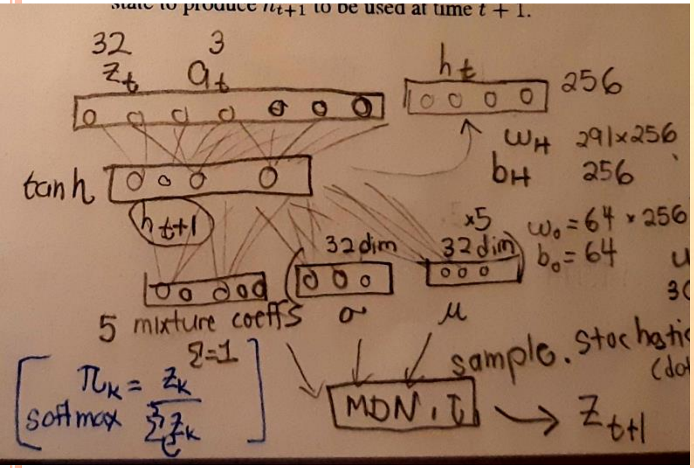
MDN for handwriting generation

MDN-RNN



② MDN-RNN
256 + 291(256)
+5 + [5 (64×256)] + 5×256 =
+5×64
158533?
hidden
h_t
RNN
h_{t+1}
a_t z_t
input
32 3
z_t a_t
tanh
h_{t+1}
h_t
256
W_H 291×256
b_H 256
×5
W_o = 64 × 256] ×5
b_o = 64
32 dim
32 dim
unrolled over 300 time steps?
5 mixture coeffs
Σ=1
σ
μ
sample. stochastic, complex env.
$\pi_k = \frac{z_k}{\sum z_k}$
softmax
MDN, τ
z_{t+1}

MDN-RNN



② MDN - RNN — why? stochastic complex environment (dot the i's, fireballs) different modes possible

★ h_{t+1} affects the predicted z_{t+1}

- minimize the rnn_r_loss(y_true, y_pred)

rnn_r_loss = average over rollout length (300 time steps) of
-log(tf_normal(y_true, mu, sigma, pi))

where

pi, mu, sigma = get_mixture_coef(y_pred)

minimize Loss = $\underbrace{-\log \text{likelihood}}$

maximize likelihood $P(\vec{z}_{t+1}\ \text{really} \mid [\vec{z}_t, \vec{a}_t, \vec{h}_t])$ — $\vec{x}$ predicted distribution

$$= \sum_{k=0}^{5} \pi_k(\vec{x}) \left[\phi(\vec{z}_{really}, \mu_k \text{ from prediction}, \sigma_k \text{ predicted}) \right]$$

weighted sum of 5 Gaussians

Product over $i = 1 \sim 32$ dimensions $\frac{1}{\sqrt{2\pi}\,\sigma_{k,i}} e^{-\frac{(z_{really,i} - \mu_{k,i})^2}{2\sigma_{k,i}^2}}$ (kth gaussian, ith dim)

- RmsProp (normalized) type of gradient descent

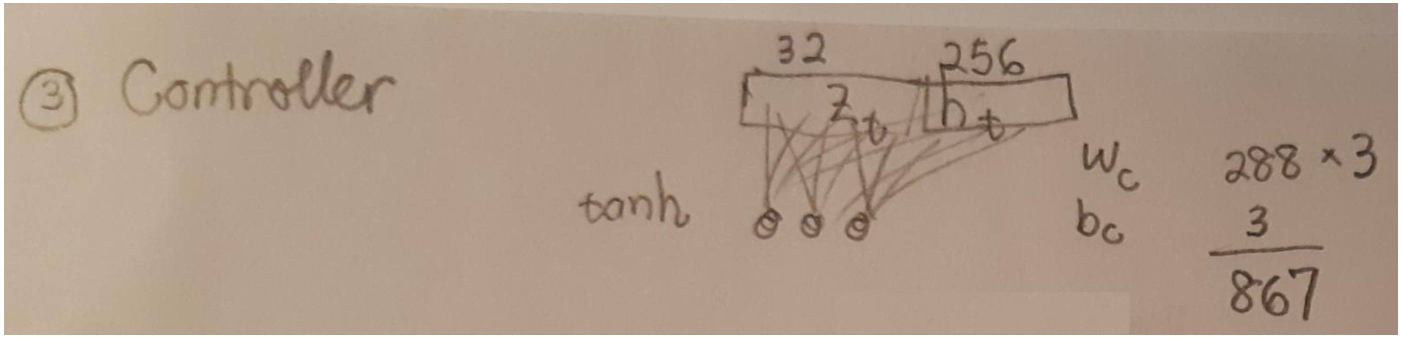
- Train on 1600 episodes (races), validate on 400
20 epochs (batches of 32 for gradient descent)

MDN-RNN

- More information
  - http://blog.otoro.net/2015/12/28/recurrent-net-dreams-up-fake-chinese-characters-in-vector-format-with-tensorflow/
  - http://blog.otoro.net/2015/12/12/handwriting-generation-demo-in-tensorflow/
  - http://blog.otoro.net/2015/11/24/mixture-density-networks-with-tensorflow/ *
  - * Alex Graves 2013 paper on *Generating Sequences with Recurrent Neural Networks* https://arxiv.org/abs/1308.0850
  - * Bishop's 1994 paper

CONTROLLER

- Vanilla neural network
- Input: 32 (z) + 256 (h)
- Output: values for the 3 actions (steer -1~1, accelerate 0~1, brake 0~1)
- (32+256) *3 = 867 parameters in C-Model

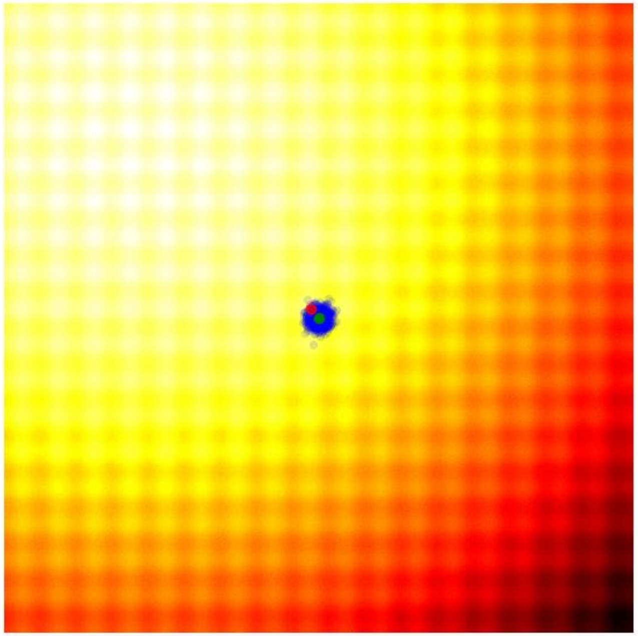


$$a_t = W_c \, [z_t \; h_t] \; + b_c$$

Controller: CMA-ES
Covariance Matrix Adaptation - Evolution Strategy

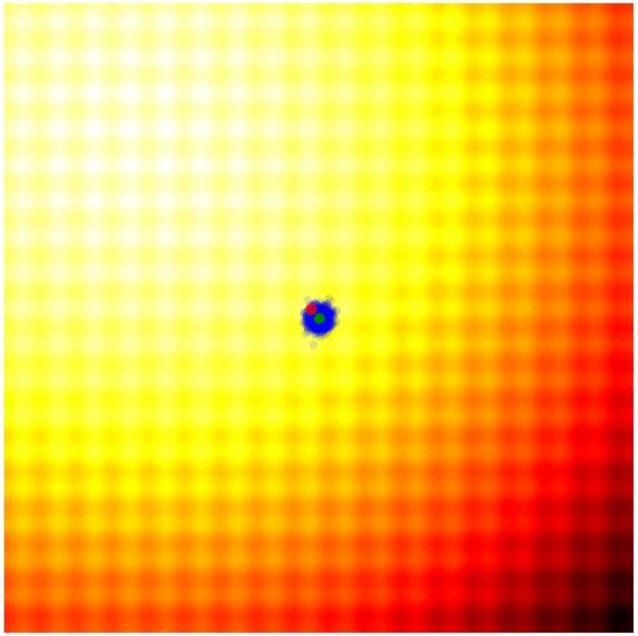
- Credit Assignment problem: The final reward is at the end of many time-steps.
  - What part of the sequence of actions resulted in the final reward? It is very unclear.
  - Traditional RL assigns a reward (decaying backward in time) for every time's action. Then it backpropagates the gradient through all the actions.
  - The Evolution Strategy does away with the gradient. It uses 'natural selection' to find the controller (car/agent) parameters so that the best car emerges that gives a high final reward.
  - ES is only useful for < 1000 parameters. Here, the C-model is 867 parameters. Computation expensive.

Controller: CMA-ES
Covariance Matrix Adaptation - Evolution Strategy



- 2D parameter space (here, it is 867 parameters = 1 car)
- Each dot is a car. There are 64 cars (population size) in each generation.
- Each car is run through 16 races. Its total reward is evaluated.
- For the best 25% of cars (purple dots), we calculate their average 867-dim vector (red dot).
- However, the diagonal covariance matrix is how much the best cars are spread away from the average of the **total** population. A wider net is cast when the best solutions are far way (red best average is far from green total average), and a smaller net is cast when the close.
- The next generation of 64 cars is sampled from N(mu, sigma)

Controller: CMA-ES
Covariance Matrix Adaptation - Evolution Strategy



$$\sigma_x^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})^2,$$

$$\sigma_y^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (y_i - \mu_y^{(g)})^2,$$

$$\sigma_{xy}^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})(y_i - \mu_y^{(g)}).$$

$$\mu_x^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} x_i,$$

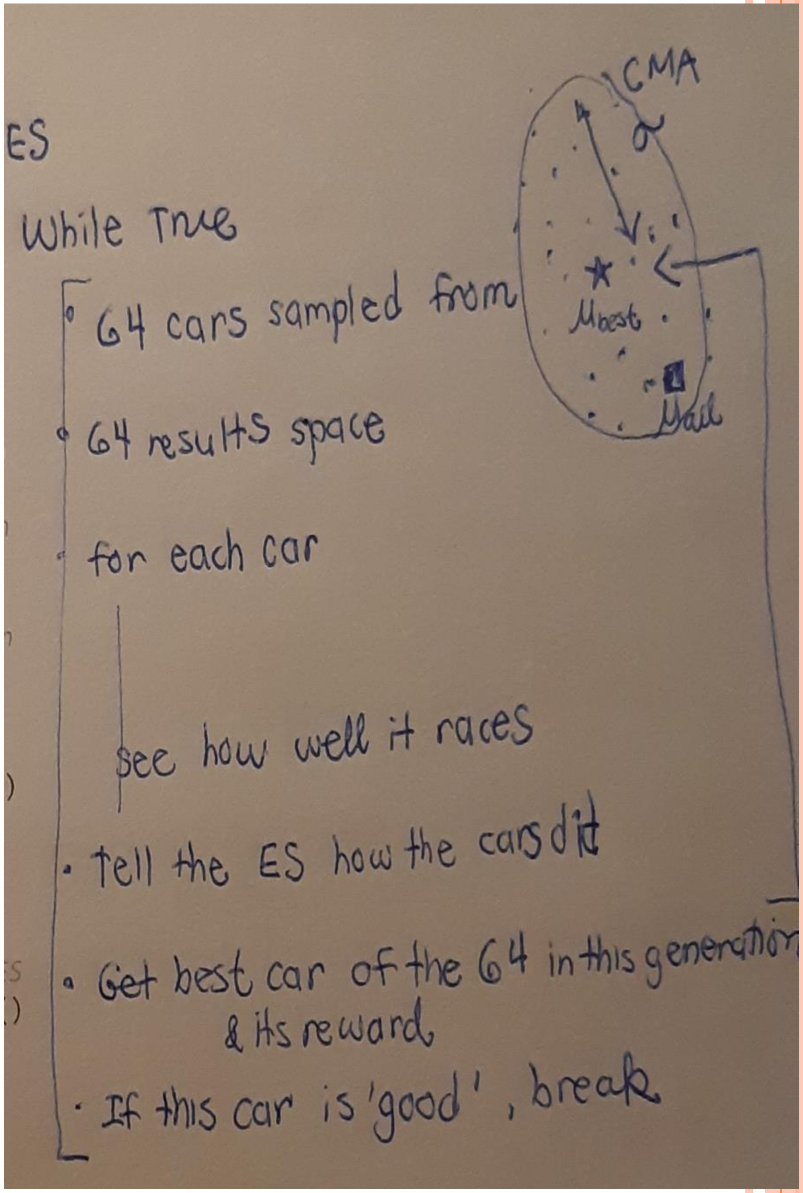
$$\mu_y^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} y_i.$$

CMA-ES

an OpenAI Gym environment, where we only care about the cumulative reward:

```python
def rollout(agent, env):
  obs = env.reset()
  done = False
  total_reward = 0
  while not done:
    a = agent.get_action(obs)
    obs, reward, done = env.step(a)
    total_reward += reward
  return total_reward
```

```python
env = gym.make('worlddomination-v0')

# use our favourite ES
solver = EvolutionStrategy()

while True:

  # ask the ES to give set of params
  solutions = solver.ask()

  # create array to hold the results
  fitlist = np.zeros(solver.popsize)

  # evaluate for each given solution
  for i in range(solver.popsize):

    # init the agent with a solution
    agent = Agent(solutions[i])

    # rollout env with this agent
    fitlist[i] = rollout(agent, env)

  # give scores results back to ES
  solver.tell(fitness_list)

  # get best param & fitness from ES
  bestsol, bestfit = solver.result()

  # see if our task is solved
  if bestfit > MY_REQUIREMENT:
    break
```



CMA-ES

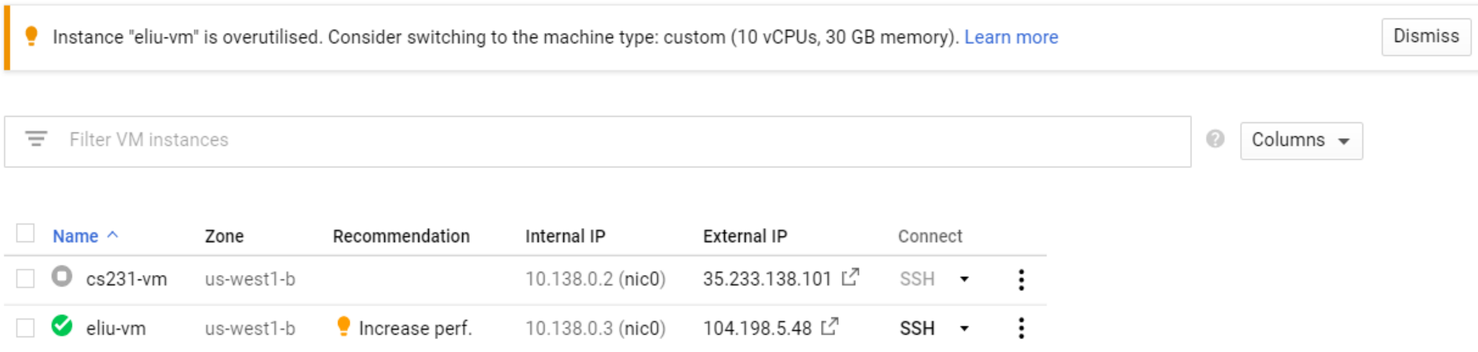
- Further Reading
  - http://blog.otoro.net/2017/10/29/visual-evolution-strategies/
  - http://blog.otoro.net/2017/11/12/evolving-stable-strategies/

Attempt to Implement the Project

- Big Data
- Code
- Use Google Cloud

- Main focus:
  - Learning about the algorithms
  - Gathering idea of the logic flow in the 30 pages of code
  - Implementing the project on remote server
    - How to download the files?
    - How to use Linux Ubuntu commands
    - Hardware and time limitations, especially for CMA-ES (how to tune hyperparameters?)
  - Results are preliminary but show some promise

1) Set Up the Environment



- http://cs231n.github.io/gce-tutorial/
- Google Cloud homepage
- Remember to turn off the instance
- David Ha's specs:
  - Ubuntu 16.04, 64 vCPU, ? GB RAM
- David Foster's specs:
  - Ubuntu 16.04, 16 vCPU, 67.5 GB RAM
- My specs:
  - Ubuntu 16.04, 8 vCPU, 40 GB disk
  - make sure to use Ubuntu (not Debian Linux)

1) Set

eliu-vm

Remote access

SSH | Connect to serial console

Enable connecting to serial ports

Logs

Stackdriver Logging

Serial port 1 (console)

More

Machine type

n1-standard-8 (8 vCPUs, 30 GB memory)

CPU platform

Unknown CPU Platform

Zone

us-west1-b

Labels

Network interfaces

Name	Network	Subnetwork	Primary internal IP	Alias IP ranges	External IP	Network Tier	IP forwarding	Network details
nic0	default	default	10.138.0.3	–	Ephemeral	Premium	Off	View details

Public DNS PTR Record

None

Firewalls

- [x] Allow HTTP traffic
- [x] Allow HTTPS traffic

Network tags

http-server, https-server

Deletion protection

- [ ] Enable deletion protection

When deletion protection is enabled, instance cannot be deleted. Learn more

Boot disk and local disks

Name	Size (GB)	Type	Mode
eliu-vm	40	Standard persistent disk	Boot, read/write

- [ ] Delete boot disk when instance is deleted

Additional disks

None

Availability policies

Preemptibility	Off (recommended)
Automatic restart	On (recommended)
On host maintenance	Migrate VM instance (recommended)

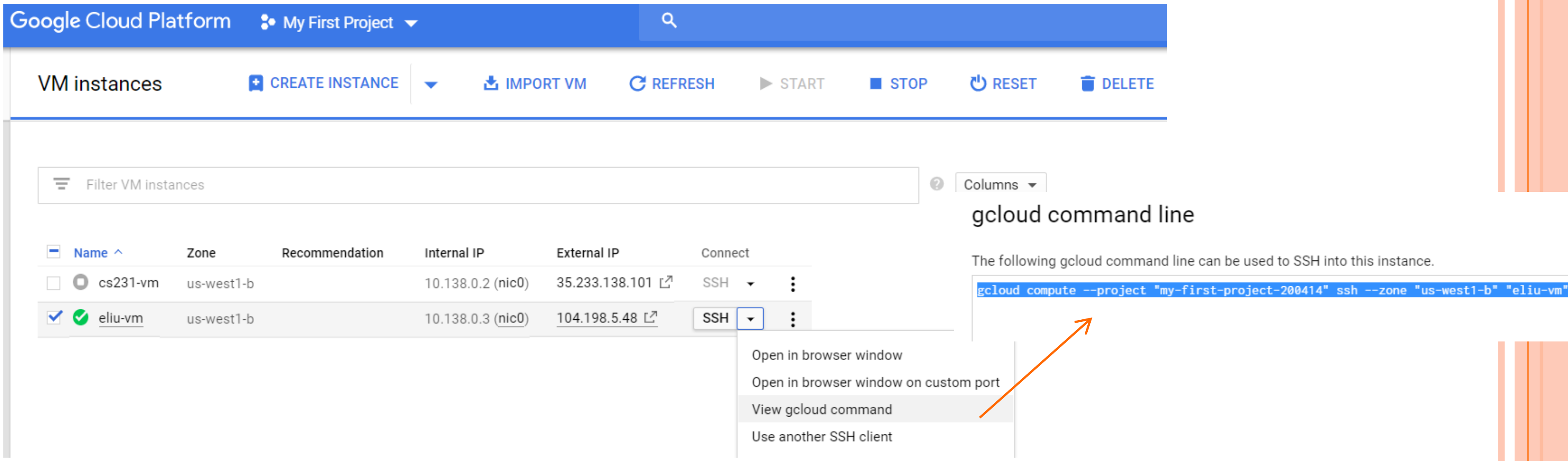
Custom metadata

1) Set up the Environment

- Limitations of ‘free trial’
  - 1 year, US$300 credits
  - My instance ~US$195/month if turned on all day
  - Need credit card to sign up but not charged until upgrade
  - maximum 8 vCPUs
  - no GPU (David Ha’s paper says a GPU makes the 2D images process faster)
  - no TPU
  - no SSD persistent disk
  - no Cloud Storage (like transferring instance files into Google Drive)
  - only HDD

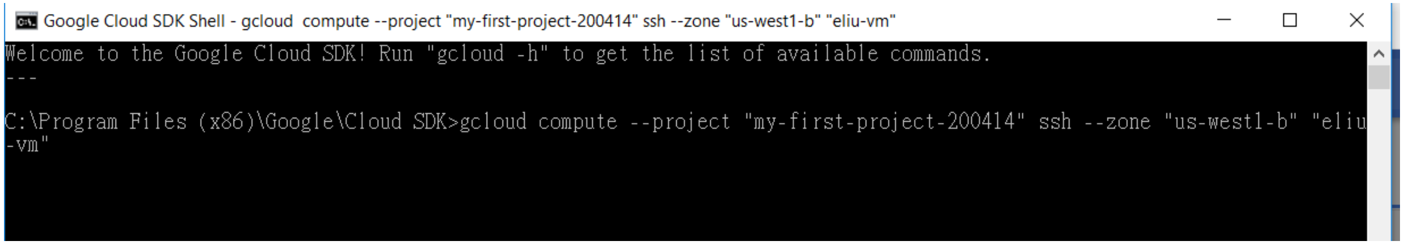
2) How to Shell Into Remote Server

- Install Google Cloud SDK Google Cloud SDK https://cloud.google.com/sdk/docs/

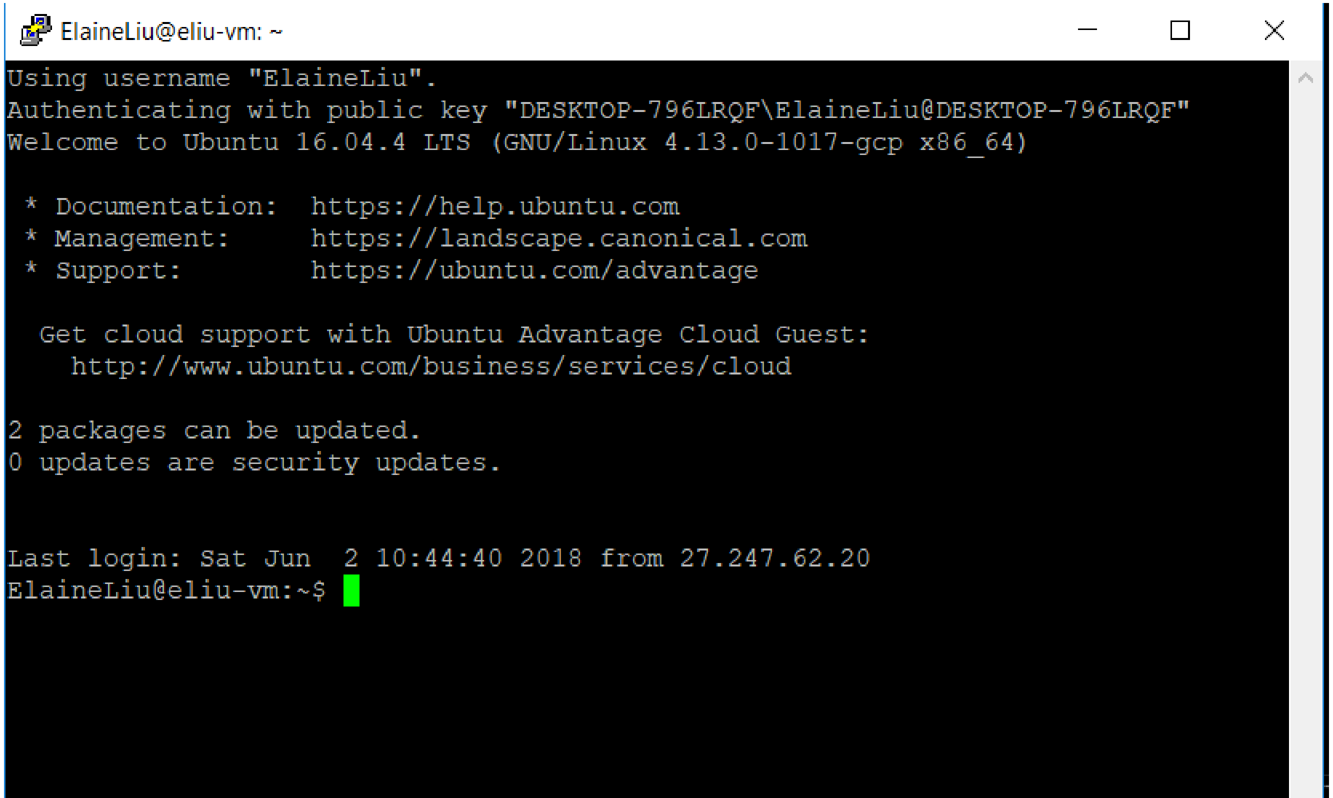


2) How to Shell Into Remote Server

Paste it in the SDK terminal



Another (PuTTy) window will appear



2) How to Shell Into Remote Server

- Permissions problems? **`sudo -i`**

- Other useful commands
  - **`cd /home`** this will change directory to the home working directory
  - **`ls -l --block-size=M`** lists the files and folders of current directory, and details of megabytes
  - **`../`** go up one directory
  - **`mkdir hw`** I created a hw folder in my home directory

3. Clone the *World Models* github code

- First cd into the directory where you want to install the WorldModels code. Then:
- **`git clone https://github.com/AppliedDataSciencePartners/WorldModels.git`**

```
root@eliu-vm:/home/hw/WorldModels# ls
01_generate_data.py      controller         es.py              rnn
02_train_vae.py          controller208      LICENSE            vae_eliu
03_generate_rnn_data.py  controller_first   log                videos
04_train_rnn.py          custom_envs        model.py           videos208
05_train_controller.py   data_eliu          __pycache__        videos_first
config.py                env.py             README.md          worldmodels
config.pyc               env.pyc            requirements.txt
```

4. Create a Python Virtual Environment

4. Create a Python virtual environment.

This is like a self-contained workspace where all the libraries and dependencies are installed for this project.

```
sudo apt-get install python-pip
sudo pip install virtualenv
sudo pip install virtualenvwrapper
export WORKON_HOME=~/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
mkvirtualenv --python=/usr/bin/python3 worldmodels
```

```
root@eliu-vm:/home/hw/WorldModels# export WORKON_HOME=~/.virtualenvs
root@eliu-vm:/home/hw/WorldModels# source /usr/local/bin/virtualenvwrapper.sh
root@eliu-vm:/home/hw/WorldModels# workon worldmodels
(worldmodels) root@eliu-vm:/home/hw/WorldModels#
```

- To reactivate,

```
export WORKON_HOME=~/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
workon worldmodels
```

5. Install packages

After you are in the virtual environment:

```
sudo apt-get install cmake swig python3-dev zlib1g-dev python-opengl mpich
xvfb xserver-xephyr vnc4server
```

6. Install libraries

```
cd WorldModels
pip install -r requirements.txt
```

The requirements.txt lists out all the very many libraries that need to be installed.

7. Generate the 600,000 Frames of Data

TIMING: This step took about 20 minutes per batch * 10 batches ~ 200 minutes.

It actually took about 2.5 hours.

COMMAND-LINE OUTPUT

```
(worldmodels) root@eliu-vm:/home/hw/WorldModels# xvfb-run -a -s "-screen 0 1400x900x24" python
01_generate_data.py car_racing --total_episodes 2000 --start_batch 0 --time_steps 300
Generating data for env car_racing
WARN: gym.spaces.Box autodetected dtype as <class 'numpy.float32'>. Please provide explicit dtype.
-----
Batch 0 Episode 0 finished after 301 timesteps
Current dataset contains 300 observations
-----
Batch 0 Episode 1 finished after 301 timesteps
Current dataset contains 600 observations
-----
```

7. Generate the 600,000 frames of Data

```
-----
Batch 9 Episode 199 finished after 301 timesteps
Current dataset contains 60000 observations
Saving dataset for batch 9
(worldmodels) root@eliu-vm:/home/hw/WorldModels#
```

Here is the data that I currently have in the data folder. This step creates the action_data_car_racing_*.npy and obs_data_car_racing_*.npy (The other numpy files are from later steps below).

```
(worldmodels) root@eliu-vm:/home/hw/WorldModels/data_eliu# ls
action_data_car_racing_0.npy  obs_data_car_racing_6.npy  rnn_input_9.npy
action_data_car_racing_1.npy  obs_data_car_racing_7.npy  rnn_output_0.npy
action_data_car_racing_2.npy  obs_data_car_racing_8.npy  rnn_output_1.npy
action_data_car_racing_3.npy  obs_data_car_racing_9.npy  rnn_output_2.npy
action_data_car_racing_4.npy  rnn_input_0.npy            rnn_output_3.npy
action_data_car_racing_5.npy  rnn_input_1.npy            rnn_output_4.npy
action_data_car_racing_6.npy  rnn_input_2.npy            rnn_output_5.npy
action_data_car_racing_7.npy  rnn_input_3.npy            rnn_output_6.npy
action_data_car_racing_8.npy  rnn_input_4.npy            rnn_output_7.npy
action_data_car_racing_9.npy  rnn_input_5.npy            rnn_output_8.npy
obs_data_car_racing_3.npy     rnn_input_6.npy            rnn_output_9.npy
obs_data_car_racing_4.npy     rnn_input_7.npy
obs_data_car_racing_5.npy     rnn_input_8.npy
```

- Not attached because:
  - > 20 GB
  - Had trouble with methods for downloading files

8. Train the VAE

- **TIMING**

It took around 7 minutes per batch * 10 batches = 70 minutes to run train the VAE.

- **python 02_train_vae.py --start_batch 0 --max_batch 9 --new_model**

```
Building batch 0...
Found car_racing...current data size = 200 episodes
Train on 48000 samples, validate on 12000 samples
Epoch 1/1
```

- **48000/48000 [==============================] - 455s 9ms/step - loss: 0.1604 - vae_r_loss: 0.1413 - vae_kl_loss: 0.0191 - val_loss: 0.1241 - val_vae_r_loss: 0.0994 - val_vae_kl_loss: 0.0246**

8. Train the VAE

batch	loss	vae_r_loss	vae_kl_loss	val_loss	val_vae_r_loss	val_vae_kl_loss
0	0.1604	0.1413	0.0191	0.1241	0.0994	0.0246
1	0.1186	0.0975	0.0211	0.1469	0.0949	0.052
2	0.116	0.0931	0.0229	0.0983	0.0834	0.0149
3	0.1113	0.091	0.0203	0.1121	0.088	0.0241
4	0.1234	0.0957	277	0.1103	0.093	0.0173
5	0.1205	0.0946	0.026	0.1287	0.1012	0.0275
6	0.1188	0.0926	0.0263	0.1204	0.1034	0.017
7	0.1168	0.0915	0.0253	0.1141	0.0939	0.0202
8	0.1168	0.0914	0.0254	0.1212	0.0947	0.0265
9	62039490.9	0.0881	62039490.81	0.0995	0.0821	0.0174

- Within a batch, the loss decreases as the samples iterate toward 48000.
- total loss = vae_r_loss (reconstruction) + vae_kl_loss
- As it trains over sets of 60000 images, the loss tends to decrease.
- Strangely, batch 9 had an anomaly. NaN?

9. Format data for RNN training

- **TIMING**

This step took 17 minutes.

- Each of 599,999 frames' [z, a] (input) and next z (correct output)

`python 03_generate_rnn_data.py --start_batch 0 --max_batch 9`

```
(worldmodels) root@eliu-vm:/home/hw/WorldModels/data_eliu# ls
action_data_car_racing_0.npy  obs_data_car_racing_6.npy  rnn_input_9.npy
action_data_car_racing_1.npy  obs_data_car_racing_7.npy  rnn_output_0.npy
action_data_car_racing_2.npy  obs_data_car_racing_8.npy  rnn_output_1.npy
action_data_car_racing_3.npy  obs_data_car_racing_9.npy  rnn_output_2.npy
action_data_car_racing_4.npy  rnn_input_0.npy            rnn_output_3.npy
action_data_car_racing_5.npy  rnn_input_1.npy            rnn_output_4.npy
action_data_car_racing_6.npy  rnn_input_2.npy            rnn_output_5.npy
action_data_car_racing_7.npy  rnn_input_3.npy            rnn_output_6.npy
action_data_car_racing_8.npy  rnn_input_4.npy            rnn_output_7.npy
action_data_car_racing_9.npy  rnn_input_5.npy            rnn_output_8.npy
obs_data_car_racing_3.npy     rnn_input_6.npy            rnn_output_9.npy
obs_data_car_racing_4.npy     rnn_input_7.npy
obs_data_car_racing_5.npy     rnn_input_8.npy
```

10. Train the MDN-RNN

- **`python 04_train_rnn.py --start_batch 0 --max_batch 9 --new_model`**
- Input [z,a]
- Predict output for next z
- Minimize loss (negative log likelihood of predicting the true next z using the distribution created from the current [z, a, h]) using RmsProp gradient descent
- Train on 1600 races, validate on 400 races. Training batches of 32 for the gradient descent.
- In the Command-line Output, we see that the training stops in 15 epochs. The loss is based on the rnn_r_loss (reconstruction loss), but the KL loss is also shown. The loss continually decreases over epochs, and as expected, the validation loss is slightly worse than training. The training stops early at 15 epochs.

10. Train the MDN-RNN

- **TIMING**

This step took about 12 minutes for epochs 1 through 15.

```
(worldmodels) root@eliu-vm:/home/hw/WorldModels#
  python 04_train_rnn.py --start_batch 0 --max_batch
  9 --new_model
```

10. Train the MDN-RNN

Epoch	loss	rnn_r_loss	rnn_kl_loss	val_loss	val_rnn_r_loss	val_rnn_kl_loss
1	1.4033	same as left	0.0205	1.3997	same as left	0.0189
2	1.3986		0.0206	1.3979		0.0189
3	1.3972		0.0211	1.397		0.0209
4	1.3967		0.0215	1.396		0.0218
5	1.3962		0.0221	1.3961		0.0214
6	1.3958		0.0224	1.396		0.0226
7	1.3957		0.0227	1.3962		0.0215
8	1.3953		0.0234	1.3972		0.0221
9	1.395		0.024	1.3959		0.0243
10	1.395		0.0248	1.3956		0.025
11	1.3947		0.0257	1.3968		0.0242
12	1.3946		0.0266	1.3961		0.0254
13	1.3943		0.0276	1.3963		0.0278
14	1.3942		0.0287	1.3957		0.0284
15	1.3941		0.0298	1.3972		0.0291

- The loss decreases over the epochs and early stopping at 15 epochs when the delta for loss is only 0.0001

11. Train the Controller

```
xvfb-run -s "-screen 0 1400x900x24" python
  05_train_controller.py car_racing --num_worker 64? --
  num_worker_trial 1? --num_episode 16 --max_length
  1000 --eval_steps 25  # David Ha's settings

xvfb-run -s "-screen 0 1400x900x24" python
  05_train_controller.py car_racing --num_worker 16 --
  num_worker_trial 2 --num_episode 4 --max_length 1000
  --eval_steps 25  # David Foster's settings

xvfb-run -s "-screen 0 1400x900x24" python
  05_train_controller.py car_racing --num_worker 8 --
  num_worker_trial 4 --num_episode 4 --max_length 1000
  --eval_steps 25  # my settings, first try

xvfb-run -s "-screen 0 1400x900x24" python
  05_train_controller.py car_racing --num_worker 8 --
  num_worker_trial 2 --num_episode 2 --max_length 1000
  --eval_steps 25  # my settings, second try
```

11. Train the Controller

- Note: go into the code to change it to only 200 generations instead of 2000 generations. Use **`vim`**, **`i`** to go into insert mode, *escape* to get out of insert mode, **`:wq`** to save and quit the editor.
- It took an extraordinary amount of time to run with limited hardware. I found that any time we need to use xvfb-run (that is, using graphics), that part of the code runs very slowly. This may be because the Google Cloud Platform free trial does not have GPU.
  - First try: >10 minutes/generation. 6 hours for 31 generations.
  - 2nd try: > 2 minutes/generation. Over 10 hours for 208 generations.
- The parameters (how many races per car? How many cars to populate one generation? How many generations?) require hand-tuning, as I found out from my first and second tries.

11. Train the Controller (CMA-ES)

- Try 1: 8 cores, 4 cars per core = 32 cars per generation. Each car is tested for 4 races.
- **(generation, seconds elapsed, avg_reward of 32 cars increases in yellow box, worst performance, best performance,)**

```
('car_racing', (1, 631, -59.44, -75.31, -43.54, 7.95, 0.49479, 1000.0, 1000))
completed episode 1 of 1
('car_racing', (2, 1240, -59.0, -70.24, -44.1, 6.67, 0.49004, 1000.0, 1000))
('car_racing', (3, 1853, -53.27, -73.25, -35.23, 9.4, 0.48566, 1000.0, 1000))
('car_racing', (4, 2455, -50.63, -69.14, -30.02, 9.87, 0.48153, 1000.0, 1000))
('car_racing', (11, 6717, -50.38, -72.85, -27.31, 10.81, 0.45796, 1000.0, 1000))
('car_racing', (12, 7341, -50.57, -66.9, -22.84, 9.78, 0.45524, 1000.0, 1000))
('car_racing', (13, 7949, -41.17, -69.59, -13.28, 11.78, 0.45263, 1000.0, 1000))
('car_racing', (14, 8557, -49.45, -78.05, -23.39, 16.28, 0.45017, 1000.0, 1000))
('car_racing', (15, 9169, -46.26, -76.59, -22.34, 15.26, 0.44782, 1000.0, 1000))
('car_racing', (16, 9780, -42.51, -69.17, -25.16, 9.9, 0.44557, 1000.0, 1000))
('car_racing', (17, 10381, -45.94, -70.63, -22.72, 13.38, 0.44345, 1000.0, 1000))
('car_racing', (18, 10985, -44.25, -65.45, -19.04, 10.61, 0.4414, 1000.0, 1000))
('car_racing', (19, 11606, -46.36, -65.82, -24.96, 11.77, 0.43939, 1000.0, 1000))
('car_racing', (20, 12222, -43.42, -72.3, -24.44, 12.81, 0.43743, 1000.0, 1000))
('car_racing', (21, 12839, -44.53, -70.36, -22.84, 10.37, 0.43556, 1000.0, 1000))
('car_racing', (22, 13459, -44.1, -70.79, -22.99, 10.06, 0.43374, 1000.0, 1000))
('car_racing', (23, 14076, -45.15, -70.26, -25.85, 11.87, 0.43201, 1000.0, 1000))
```

standard deviation of population

standard deviation used for CMA-ES aka casting the net wide or narrow

none of the cars finished the races before the allotted time

11. Train the Controller

- Try 2: 8 cores, 2 cars per core (population = 16 cars per generation), each car runs 2 races.
- I stopped the training after 208 generations

- Every 25 steps, there is an evaluation (see next page)
- It checks if the best car in the current generation is better or worse (how much improvement) than the best car in history. If the current generation's best car is better, then it is set as the new best car in history.

CMA-ES Try #2

"improvement", t, improvement, "curr", reward_eval, "prev", prev_best_reward_eval, "best", best_reward_eval)

```
('car_racing', (1, 163, -57.46, -73.06, -40.3, 8.77, 0.49662, 1000.0, 1000))
('car_racing', (25, 3973, -50.21, -76.04, -23.85, 12.5, 0.44603, 1000.0, 1000))
('improvement', 25, -178.84756875, 'curr', -236.30756875, 'prev', -57.46, 'best', -236.30756875)
('car_racing', (50, 8468, -47.73, -68.49, -12.61, 15.33, 0.41829, 1000.0, 1000))
('improvement', 50, -2.2687999999999704, 'curr', -238.57636874999997, 'prev', -236.30756875, 'best',
-236.30756875)
('car_racing', (75, 12993, -42.56, -69.51, -21.25, 11.71, 0.40128, 1000.0, 1000))
('improvement', 75, 4.120056250000005, 'curr', -232.1875125, 'prev', -236.30756875, 'best', -232.1875125)
('car_racing', (100, 17525, -41.02, -71.45, -1.87, 19.46, 0.39043, 1000.0, 1000))
('improvement', 100, 13.867756249999985, 'curr', -218.31975625, 'prev', -232.1875125, 'best', -218.31975625)
('car_racing', (125, 22012, -46.4, -81.11, -22.13, 16.93, 0.38328, 1000.0, 1000))
('improvement', 125, -8.613587499999994, 'curr', -226.93334375, 'prev', -218.31975625, 'best', -218.31975625)
('car_racing', (150, 26495, -45.57, -75.76, -26.36, 14.99, 0.37946, 1000.0, 1000))
('improvement', 150, -18.758350000000007, 'curr', -237.07810625000002, 'prev', -218.31975625, 'best',
-218.31975625)
('car_racing', (175, 30982, -45.0, -72.37, -23.63, 12.8, 0.37749, 1000.0, 1000))
('improvement', 175, -14.13615624999997, 'curr', -232.45591249999998, 'prev', -218.31975625, 'best',
-218.31975625)
('car_racing', (200, 35456, -45.05, -74.28, -30.19, 10.89, 0.37592, 1000.0, 1000))
('improvement', 200, -15.498643749999985, 'curr', -233.8184, 'prev', -218.31975625, 'best', -218.31975625)
```

12. Test the best car

```
xvfb-run -s "-screen 0 1400x900x24" python model.py
  car_racing --filename
  ./controller/car_racing.cma.4.32.best.json --
  render_mode --record_video
```

Use car_racing.cma.2.16.best.json for try #2

TRY #1

- The demo shows that the car can turn left, but needs to deal with sharp corners better.
- It went off the track, but managed to find its way back.

TRY #2 was almost random

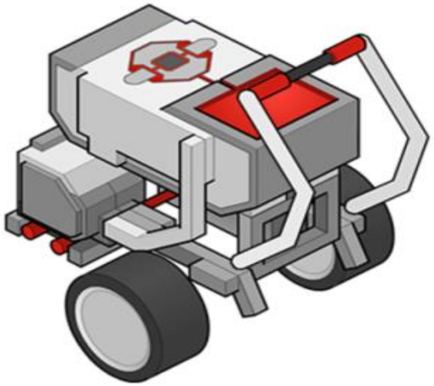
This indicates lack of diversity in the evolution (need bigger population size and more test races per car)

Future Extensions

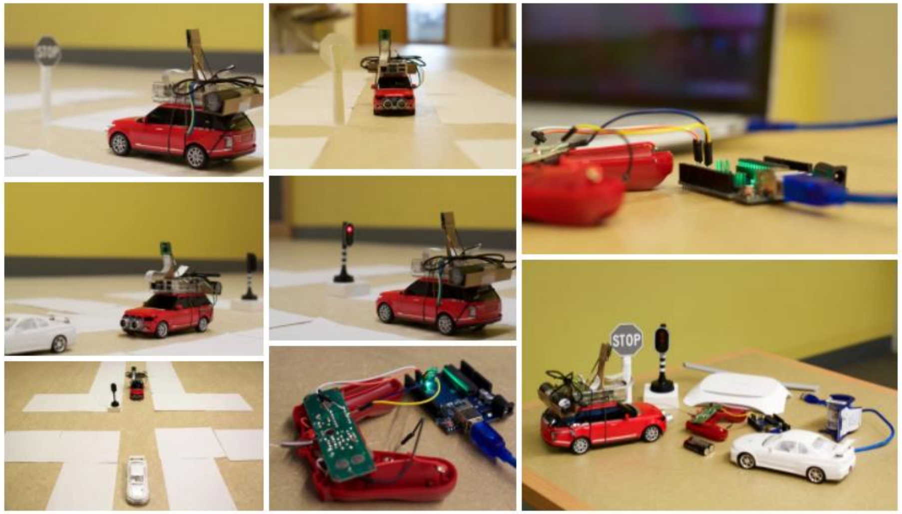
- Dreamed-up Environment used to train the car
- Temperature of the RNN and effect on policy
- Evolutionary strategy

FUTURE EXTENSIONS

- Apply AI steering to autonomous Lego EV3

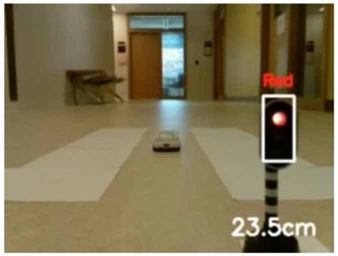


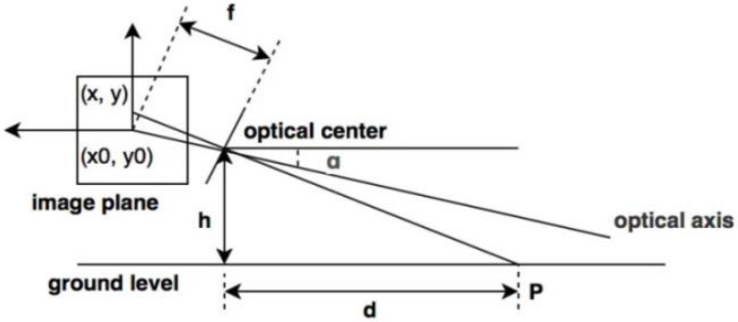
- First step: Run through an open-source example to figure out how to modify it so that the hardware interface is set up



Future Extensions

- The nice example online is open-source
  - Steering control (up, down, left, right arrow keys for steering left, right, forward, reverse) – uses a simple multi-layer perceptron model, the project trains the neural network from scratch
  - Object detection and monocular camera distance measurement to stop at sign and traffic light (Computer Vision object classifier)
  - Ultrasonic sensor to prevent front collision

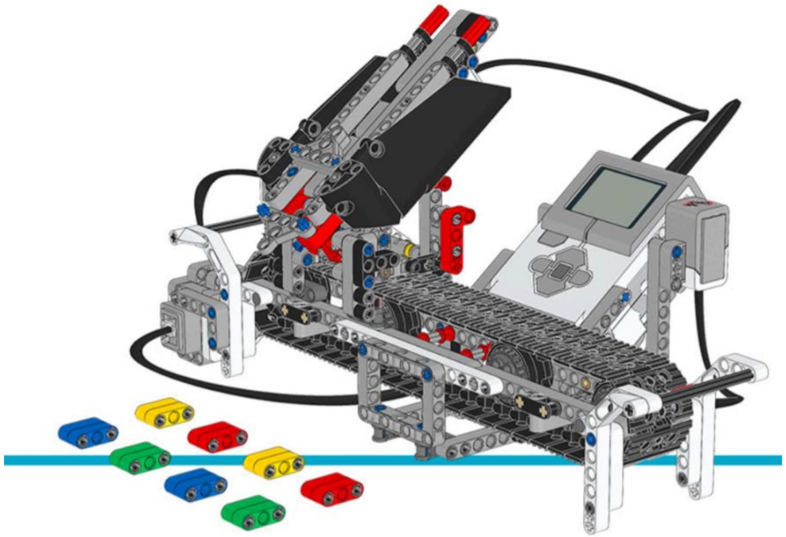




Future Extensions

It is possible to use neural networks to control the Lego EV3

- Object classification for sorting



- Q-Learning to learn to crawl

CONCLUSION

- The World Models paper introduces the flow of a large project
- Integrates 3 algorithms (VAE, MDN-RNN, CMS-ES)
- CMA-ES takes a lot of time to compute
  - Hardware limitations
  - How to fine-tune population size, races to run per car, how many generations?
  - Can the controller training be replaced by faster methods? (backpropagation? RL?)
- Extensions:
  - Upgrade hardware specs.
  - Check the VAE by reconstructing images (figure out how to download large data from the Cloud)
  - Plot loss convergence neatly
  - Run the car in its hallucinated environment
  - Figure out how to assign rewards and train the controller with the dreamed race.

Thank you for Listening!