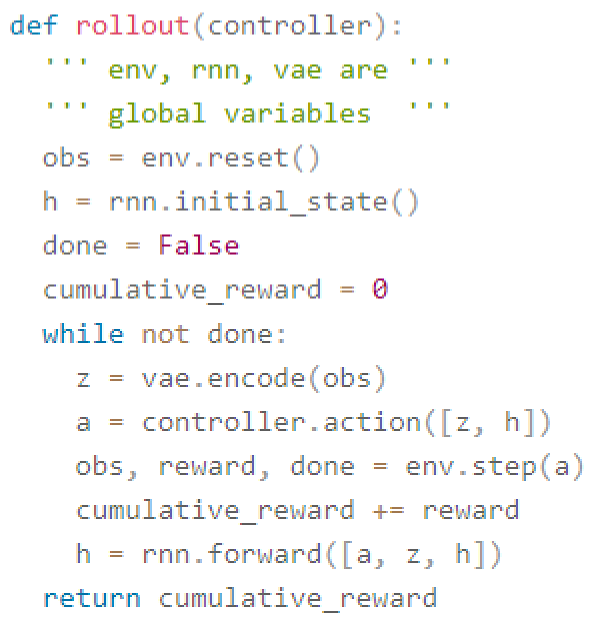
World Models:

The Agent Model is a 3-component model where 2 of the components are world models. This means that they take the images and make predictions based on reinforcement learning techniques and a controller model that makes decisions based on the world model. We will explore these models and how they work as well as how to apply them. The VAE model or Vision (V) model compresses images into small representative code and the MDN-RNN or Memory (M) model makes predictions about future codes based on historical info. First, we will train our models by gathering data from rollouts using a random policy. The pseudocode for each rollout run on a controller is defined below.



```
def rollout(controller):
  ''' env, rnn, vae are '''
  ''' global variables  '''
  obs = env.reset()
  h = rnn.initial_state()
  done = False
  cumulative_reward = 0
  while not done:
    z = vae.encode(obs)
    a = controller.action([z, h])
    obs, reward, done = env.step(a)
    cumulative_reward += reward
    h = rnn.forward([a, z, h])
  return cumulative_reward
```

VAE (V) Model:

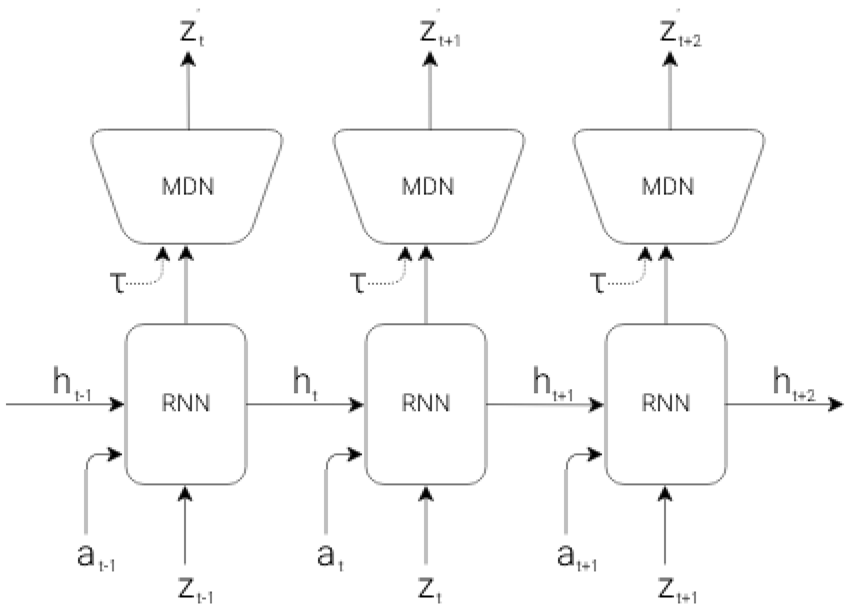
The Variational Autoencoder (VAE) model is used to reconstruct the original image by compressing what the agent sees at each time frame t into a low dimensional latent vector zt. The first step in order to set up the VAE model is to collect random rollouts of the environment. This is done by making the agent act randomly to explore the environment multiple times, and record the random actions taken. Using this dataset, train V to learn a latent space of each frame observed.

The goal of training the VAE is to encode each frame into low dimensional latent vector z while minimizing the difference between a given frame and the reconstructed version of the frame produced by the decoder from z. In the world model example, both car racing, and doom had similar procedures, but the V model in car racing was used to train the M model, while the V model in VisDoom was used to not only train M but convert the image collected into the latent space representation in which was used to train the model in a dream environment.

If the V model was the only world model used, then the agent would be very inaccurate since there is no optimality factor. This means that they will also take in irrelevant since unsupervised learning means that this model doesn’t know what data is useful and which isn't. However, the agent would be going in the right direction and have the right idea but not in an optimal manner. In order to make the agent optimal, the second piece of the world model is the MDN-RNN Model which we use to predict rewards.

MDN-RNN (M) Model:

The Mixture Density Network-Recurrent Neural Network (MDN-RNN) model takes the V model as well as the history (rollouts) to predict the future. This model is used alongside the V model in order for the controller model to make a decision on what is the most optimal path. The biggest difference between the V model and the M model is that the M model trains the RNN to output a probability density function p(z) rather than a deterministic prediction of z.



The image above shows that RNN with an MDN output layer outputs the parameters of a mixture of Gaussian distribution used to sample a prediction of the next latent vector z.

This approach approximates p(z) and trains the RNN to output the probability distribution of the next latent vector z_{t+1} given the current and past information (h). During sampling, there is a temperature parameter T which will control model uncertainty which is very important by making the training sample more challenging. Training on samples more difficult than the actual environment can vary in behavior. In the car-racing model, it would be better to keep the randomness more accurate to the actual environment as making a mistake would be more punishing, while in the Doom model, making it more difficult in a dream environment actually teaches the agent more to a certain extent. Making it too difficult may lead the agent to die due to misfortune/unavoidable circumstances which will confuse the agent.

Controller Model

The controller model should be very simple as it takes the information from the world models and comes up with choices. The C model should be as simple and as small as possible, and trained separately from V and M so it doesn't interfere with the solutions from the world models. Since V and M models are very intensive, C should be light, it'll also open up more ways to train C by using different evolution strategies such as the CMA-ES optimization algorithm.

C is a simple single layer linear model that maps z_t and h_t directly to action at at each time step:

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

W_c and b_c are the weight matrix and bias vector that maps the concatenated input vector $[z_t, h_t]$ to the output action vector a_t^3