Agent Embeddings: A Latent Representation for Pole-Balancing Networks

Oscar Chang
Columbia University
New York, New York
oscar.chang@columbia.edu

Robert Kwiatkowski
Columbia University
New York, New York
robert.kwiatkowski@columbia.edu

Siyuan Chen
Columbia University
New York, New York
sc3618@columbia.edu

Hod Lipson
Columbia University
New York, New York
hod.lipson@columbia.edu

ABSTRACT

We show that it is possible to reduce a high-dimensional object like a neural network agent into a low-dimensional vector representation with semantic meaning that we call *agent embeddings*, akin to word or face embeddings. This can be done by collecting examples of existing networks, vectorizing their weights, and then learning a generative model over the weight space in a supervised fashion. We investigate a pole-balancing task, Cart-Pole, as a case study and show that multiple *new* pole-balancing networks can be generated from their agent embeddings without direct access to training data from the Cart-Pole simulator. In general, the learned embedding space is helpful for mapping out the space of solutions for a given task. We observe in the case of Cart-Pole the surprising finding that good agents make different decisions despite learning similar representations, whereas bad agents make similar (bad) decisions while learning dissimilar representations. Linearly interpolating between the latent embeddings for a good agent and a bad agent yields an agent embedding that generates a network with intermediate performance, where the performance can be tuned according to the coefficient of interpolation. Linear extrapolation in the latent space also results in performance boosts, up to a point.

CCS CONCEPTS

• **General and reference** → **General conference proceedings**; • **Computer systems organization** → **Neural networks**; • **Theory of computation** → **Multi-agent reinforcement learning**;

KEYWORDS

Learning agent capabilities (agent models, communication, observation); Analysis of agent-based simulations; Simulation of complex systems; Deep learning

ACM Reference Format:
Oscar Chang, Robert Kwiatkowski, Siyuan Chen, and Hod Lipson. 2019. Agent Embeddings: A Latent Representation for Pole-Balancing Networks. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

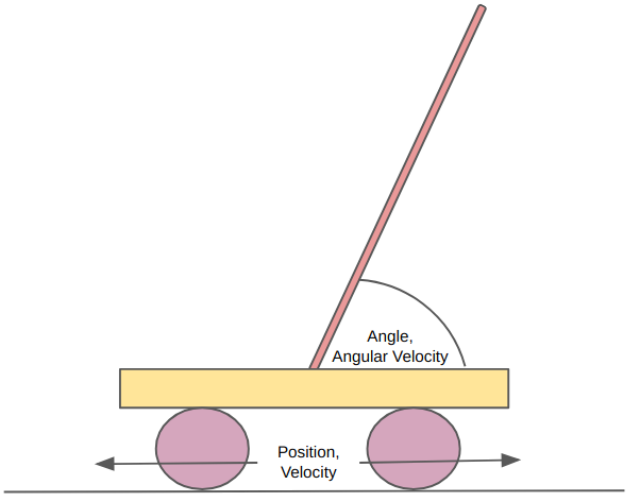

Figure 1: Cart-Pole is a game of pole-balancing

1 INTRODUCTION

Many modern artificially intelligent agents are trained with deep reinforcement learning algorithms [17, 23, 30]. But neural networks have long been criticized for being uninterpretable black boxes that cannot be relied upon in safety-critical applications [7, 38].

It is important to note, however, that human brains are uninterpretable as well. For example, we know what a face is, because our brains have evolved to detect facial features, and yet, it is nearly impossible to communicate in words what a face is. This problem is especially acute for patients with severe prosopagnosia, who have to rely on other visual cues to identify their friends and family. In fact, it is also quite difficult to communicate precisely the meaning of words. Try talking to a philosopher or a translator about what otherwise ordinary words might mean, *precisely*, and one can be sure to spark a huge debate.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

Nonetheless, it is possible to program a computer to detect faces, by reducing high-dimensional images of faces into low-dimensional vector representations with semantic meaning [27, 29]. It is also possible to perform sophisticated natural language processing tasks by representing words in a high dimensional vocabulary as low-dimensional vectors [22, 25]. Remarkably, these embeddings are amenable to simple linear arithmetic. Take the difference between the latent codes for a face with a mustache and one without a mustache, and one gets something approximating a 'mustache' vector. Famously, Mikolov et al. [22] showed 'King' - 'Queen' = 'Man' - 'Woman'.

We propose that a similar strategy can be applied to even something as high-dimensional and complicated as a deep reinforcement learning agent. Our aim is to demonstrate that neural network agents can be compressed into low-dimensional vector representations with semantic meaning, which we term *agent embeddings*. In this paper, we propose to learn agent embeddings by collecting existing examples of neural network agents, vectorizing their weights, and then learning a generative model over the weight space in a supervised fashion.

1.1 Our Contribution

As a proof of concept, we report on a series of experiments involving agent embeddings for policy gradient networks that play Cart-Pole, a game of pole-balancing.

We present three interesting findings:

(1) The embedding space learned by the generative model can be used to answer questions of convergent learning [19], i.e. how similar are different neural networks that solve the same task. To our knowledge, we are the first to investigate convergent learning in the context of reinforcement learning agents rather than image classifiers. We extend Li et al.'s work on convergent learning by proposing a new distance metric for measuring convergence between two neural networks. We observe surprisingly that good pole-balancing networks make different decisions despite learning similar representations, whereas bad pole-balancing networks make similar (bad) decisions while learning dissimilar representations.

(2) It has been demonstrated that linear structure between semantic attributes exist in the latent space of a good generative model in the domain of natural language words [22] and faces [27], among other kinds of data. We show that a similar linear structure can be learned in an embedding space for reinforcement learning agents that can be used to directly control the performance of the policy gradient network generated.

(3) We demonstrate that the generative model can be used to recover missing weights in the policy gradient network via a simple and straightforward rejection sampling method. More sophisticated methods of conditional generation are left to future work.

The rest of the paper is organized as follows: we survey the relevant literature (*Related Work*), introduce the pole-balancing task and describe how we learn agent embeddings for it (*Learning Agent Embeddings for Cart-Pole*), present the above-mentioned findings (*Experimental Results and Discussion*), discuss the shortcomings of our approach (*Limitations of Supervised Generation*), speculate on potential applications (*Potential Applications for AI*), and finally summarize the paper at the end (*Conclusion*).

2 RELATED WORK

There are four areas of research that are related to our work: interpretability, generative modeling, meta-learning, and Bayesian neural networks.

2.1 Interpretability

There has been a lot of recent interest in making reinforcement learning agents and policies interpretable. This is especially important in high-stake domains like health care and education. Verma et al. [34] proposed to learn policies in a human-readable programming language, while Dann et al. [10] proposed to learn certificates that provides guarantees on policy outcomes. Zha et al. [37] demonstrated utility in learning embeddings for action traces in path planning. Ashlock and Lee [2]'s work is very similar to ours - they proposed a tool to compare phenotypic differences between solutions found by evolutionary algorithms as a way to explore the geometry of the problem space.

One line of work that has proven useful in increasing our understanding of deep neural network models is that of convergent learning [19], which measures correlations between the weights of different neural networks with the same architecture to determine the similarity of representations learned by these different networks. Convergent learning investigations have hitherto, to our knowledge, only been done on image classifiers, but we extend them to reinforcement learning agents in this paper.

2.2 Generative Modeling

Generative modeling is the technique of learning the underlying data distribution of a training set, with the objective of generating new data points similar to those from the training set. Deep neural networks have been used to build generative models for images [27], audio [33], video [35], natural language sentences [4], DNA sequences [36], and even protein structures [1]. Complex semantic attributes can often be reduced to simple linear vectors and linear arithmetic in the latent spaces of these generative models.

The ultimate (meta) challenge for neural network based generative models is not to generate images or audio, but other neural networks. We use existing networks as meta-training points and use them to train a neural network generator that can produce new pole-balancing networks that do not then need to be further trained with training data from the Cart-Pole simulator. A key advantage of using the same learning framework for both the meta learner and the learner is that this approach could potentially be applied recursively (cue the Singularity).

2.3 Meta-Learning

The salient aspect of meta-learning that our work is connected to is the use of neural networks to generate other neural networks. This has been done before in the context of hyperparameter optimization, where one neural network is used to tune the hyperparameters of another neural network [20, 26, 31, 39]. Ha et al. proposed

the concept of a HyperNet, a neural network that generates the weights of another neural network with a differentiable function. This allows changes in the weights of the generated network to be backpropagated to the HyperNet itself. Chang and Lipson used a neural network to generate its own weights as a way to implement artificial self-replication.

2.4 Bayesian Neural Networks

Bayesian neural networks [5] maintain a probabilistic model over the weights of a neural network. In this framework, traditional optimization is viewed as finding the maximum likelihood estimate of the probabilistic model. Posterior inference in this case is typically intractable, but variational approximations can be used [15, 16, 21]. Our work involves learning a generative model over the weights of a neural network using existing examples of networks, which is philosophically akin to learning an 'empirical Bayesian' prior over the weights in a Bayesian neural network.

3 LEARNING AGENT EMBEDDINGS FOR CART-POLE

3.1 Supervised Generation

We propose to learn agent embeddings for neural networks using a two-step process we call *Supervised Generation*. First, we train a collection of neural networks of a fixed architecture to solve a particular task. Next, the weights are saved and used as training input to a generative model. This is a supervised method because we are learning the mapping from a latent distribution to the space of neural network weights by feeding input-output pairs to the model. (There are some obvious downsides to *Supervised Generation* as a method of learning agent embeddings. See the *Limitations of Supervised Generation* section for a detailed discussion.)

In this case, we trained a variational autoencoder (*CartPoleGen*) on the parameter space of a small network (*CartPoleNet*) used to play Cart-Pole.

3.2 Cart-Pole

Cart-Pole is a pole balancing task introduced by Barto et al. with a modern implementation in the OpenAI Gym [6]. It is also known as the inverted pendulum task and is a classic control problem. The agent chooses to move left or right at every time step with the objective of preventing the pole from falling over for as long as possible. We chose this task because it is easy - around 200 times easier than MNIST on one measure [18] - and hence can be solved with small neural networks.

3.3 CartPoleNet

We devised a simple policy gradient neural network we call *CartPoleNet* with exactly one hidden layer of dimension 30 (see Figure 2) using the exponential linear unit [9] as the activation function. We collected 74000 such networks by training them in the Cart-Pole simulator with varying amounts of time, hyperparameters and random seeds for over a week on a cloud computing platform. The 212-dimensional weight vectors belonging to these 74000 networks were then used as the training data for the generative model.

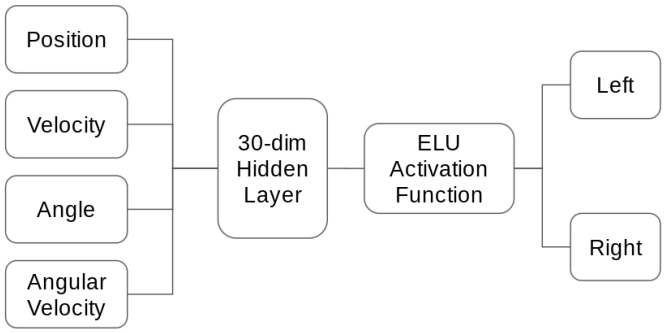

Figure 2: Architecture of CartPoleNet

A policy gradient neural network approximates the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a, \pi\right] \tag{1}$$

which is the maximum expected sum of rewards r_i discounted by γ and achieved by a policy $P(a \mid s)$ that makes an action a after observing state s. Cart-Pole assigns a reward of 1 for every step taken, and each episode terminates whenever the pole angle exceeds 12°, the position exceeds the edge of the display, or once the pole has been successfully balanced for more than 200 time steps.

At each epoch, we sample state-action pairs with an epsilon-decreasing policy and store them with their rewards in an experience replay buffer to train the neural network. Note that the neural network only takes state s as input, and its Q-value at action a is represented by the corresponding activation on the last layer. Parametrizing the Q-function with a state-action pair as input is possible but more computationally expensive because it requires $\mid A \mid$ number of forward passes where A is the action space [24].

3.4 CartPoleGen

CartPoleGen is a variational autoencoder with a diagonal Gaussian latent space of dimension 32. It contains skip connections (with concatenation not addition) and uses the exponential linear unit as the activation function as in CartPoleNet (see Figure 3).

A variational autoencoder [15] is a latent variable model with latent $\mathbf{z}$ and data $\mathbf{x}$. We assume the prior over the latent space to be the spherical Gaussian $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and the conditional likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ to be Gaussian, which we compute with a neural network decoder parametrized by θ. The true posterior $p(\mathbf{z} \mid \mathbf{x})$ is intractable in this case, but we assume that it can be approximated by a Gaussian with a diagonal covariance structure that we can compute with a neural network encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ parametrized by ϕ.

Sampling from the posterior involves reparametrizing $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ to $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to allow the gradients to backpropagate through to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

We can train the variational autoencoder by maximizing the variational lower bound on the marginal log likelihood of data point $\mathbf{x}$:

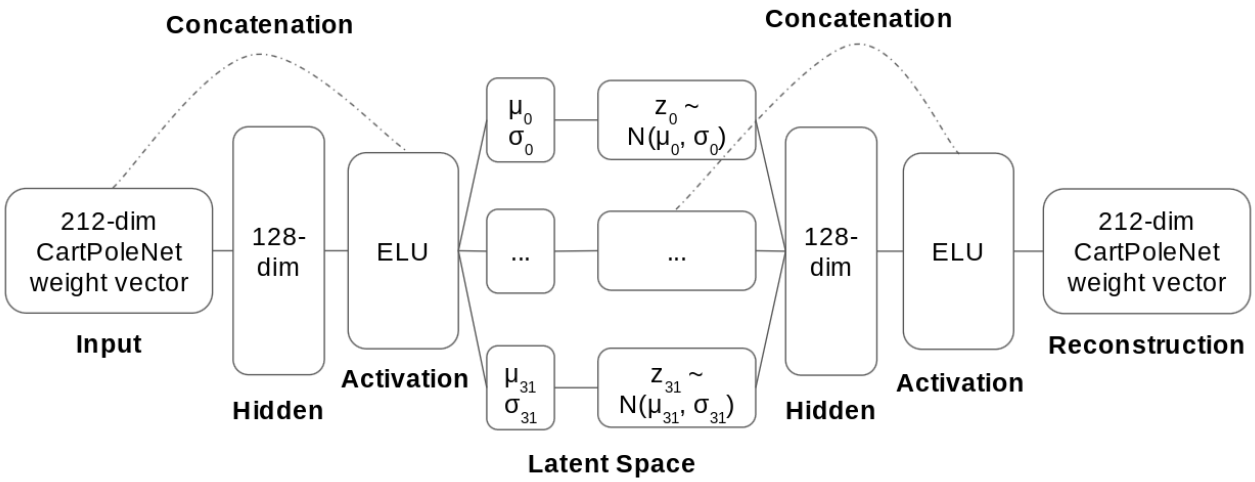

Figure 3: Architecture of CartPoleGen

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = -\mathcal{D}_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] \tag{2}$$

The Monte Carlo estimator (with latent dimension $k = 64$ and noise mini-batch of size $M = 1$) for equation (2), also known as the SGVB estimator, becomes

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \frac{1}{2}\sum_k (1 + \log \sigma_k^2 - \mu_k^2 - \sigma_k^2) + \frac{1}{M}\sum_{m=1}^{M} \log p_\theta(\mathbf{x} \mid \mathbf{z}^{(m)}) \tag{3}$$

Notice that maximizing the above lower bound involves maximizing the model's log-likelihood, which is equivalent to minimizing its negative log-likelihood. Minimizing the negative log-likelihood of a Gaussian model is equivalent to minimizing the mean squared error, which is simply the reconstruction cost in an autoencoder.

3.5 Sampling from CartPoleGen

We divided the 74000 networks into four groups depending on the network's *survival time*, which we measure as the average number of steps before the episode terminates across 100 random testing episodes. The survival time is quite a robust measure of CartPoleNet's performance; it varies ±5 at most due to the stochasticity of the Cart-Pole simulator.

We trained CartPoleGen in two settings. The first setting involves training on all 74000 networks, and then measuring the survival time of 200 new samples drawn from the posterior distribution of the variational autoencoder. The second setting involves training a separate CartPoleGen conditioned on each group with a conditional VAE setup [32]. The survival time in the second setting is also measured with 200 new samples drawn from the posterior of the conditional generative model.

The training was conducted using ADAM [14] for 20 epochs with a batch size of 10. The results are summarized in Table 1. For comparison, an agent that randomly selects actions lasts on average 22 steps, and an agent that makes the same action at every time step lasts only 9 steps. The Cart-Pole simulation ends once an agent has survived 200 steps, so it is not possible to survive longer than that.

Figure 4 shows that the CartPoleGen does not accurately capture the exact distribution of the training data, but that it does offer an approximation to it. Training on better networks tends to lead to better generated networks, with the exception of the $151 - 200$ survival time group. We surmise that this is a consequence of the unimodal variational approximation.

Curiously, CartPoleGen seems to display zero-avoiding rather than zero-forcing behavior, which show that the behavioral properties of neural network agents do not directly match their weight space properties. It is interesting that in some cases, we are able to sample new networks that dramatically outperform the original networks that were in the training set. In the conditional groups, the generated samples typically display much higher variance than is found in the training set, but this does not hold true in the combined setting.

We hypothesize that the approximation gap is partially due to the limitations of the variational autoencoder and can be narrowed with a more expressive generative model. We experimented with various other neural architectures for the encoder and decoder, but did not manage to find significant improvements. In fact, the architecture of CartPoleGen presented here approximates a similar distribution when the encoder and decoder are trained with linear layers.

We also experimented with using GANs [11, 27] as the generative model for CartPoleGen, but did not manage to successfully train them. In our experiments, the discriminator was not able to provide a good teaching signal to the generator because it managed to rapidly distinguish between the fake and real samples.

Table 1: Sampling new instances of CartPoleNet

Group	Trainset Size	(Mean, Std) of Survival Time in Trainset	(Mean, Std) of Survival Time in Generated Samples
1 – 50 steps	25608	21.8, 11.5	11.0, 9.7
51 – 100 steps	9400	69.7, 14.2	77.3, 46.5
101 – 150 steps	10103	132.6, 13.1	127.0, 55.3
151 – 200 steps	28889	184.9, 16.3	116.4, 58.6
Combined	74000	106.7, 73.3	136.7, 42.8

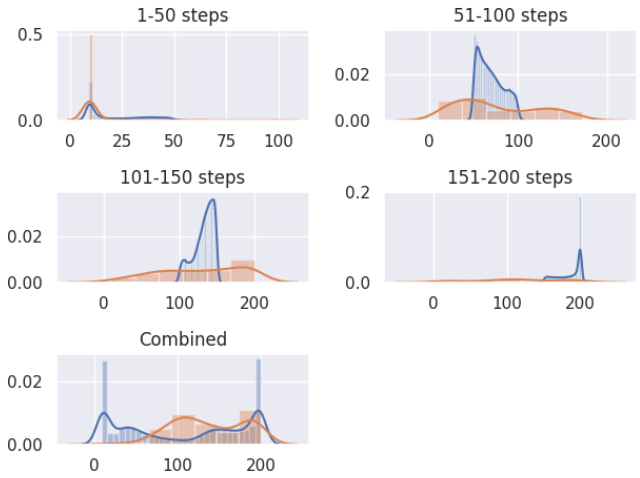

Figure 4: The figures are plotted as histograms, with KDE curves fitted on them. The x-axis denotes the survival time, and the y-axis denotes the percentage of networks with that survival time. The figures in blue represent the networks from the trainset, while the figures in orange represent the sampled networks.

4 EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we perform three experiments using the agent embeddings learned by CartPoleGen in the previous section. These experiments involve (1) deciding if different CartPoleNets of similar ability learn similar representations, (2) exploring the latent space learned by CartPoleGen, and (3) repairing missing weights in a CartPoleNet.

4.1 Convergent Learning

Li et al. posed the question of convergent learning: do different neural networks learn the same representations? In the case of convolutional neural networks used as image classifiers, they found that shallow representations that resemble Gabor-like edge detectors are reliably learned, while more semantic representations sometimes differ.

Success is usually not an accident. Prima facie, for a given complex task, it seems like there can be a million ways to fail it, but only a handful of ways to successfully solve it. We hypothesize this to be the case for Cart-Pole, but found surprisingly that the reverse was true.

Li et al. measured activations on a reference set of images from the ImageNet Large Scale Visual Recognition Challenge 2012 dataset [28], and calculated the correlation of such activations between pairs of convolutional neural networks. For CartPoleNets, the inputs are environment states in Cart-Pole, so we had to first collect a reference set of 10000 diverse states in the Cart-Pole simulator before computing CartPoleNet activations on them.

We follow the same methodology as Li et al. with the slight modification that we use the absolute value of the activations. This is because we use ELUs in CartPoleNet which have important negative activations that ReLU-based networks do not.

$$\text{Mean} : \mu_i = \mathbb{E}[|X_i|] \tag{4}$$

$$\text{Std} : \sigma_i = \sqrt{\mathbb{E}[(|X_i| - \mu_i)^2]} \tag{5}$$

$$\text{Corr} : \rho_{i,j} = \mathbb{E}[(|X_i| - \mu_i)(|X_j| - \mu_j)]/\sigma_i\sigma_j \tag{6}$$

The correlation between activations of a pair of networks can then be used to pair units from the first network with units from the second. In a bipartite matching, we assign each pair by matching units with the highest correlation, taking them out of consideration, and repeating the process until all the units have been paired. Hence, each unit belongs to exactly one pair. This can be done efficiently with the Hopcroft-Kraft algorithm [13]. In a semi-matching, we sequentially assign each unit i from the first network using the unit j from the second network with the highest correlation $\rho_{i,j}$. It is thus possible that some units will belong to multiple pairs, while others will not get paired at all.

Two networks are in some sense equivalent if we can arrive at one network by permuting the ordering of the units of the other. The *convergence distance (CD)* between two networks can hence be quantitatively measured as the distance between the bipartite matching and the semi-matching (see Equation 7). There is exactly one bipartite matching of maximum cardinality, but multiple possible semi-matchings depending on the order of assignment. We compute the convergence distance using the *canonical* semi-matching, defined as the semi-matching performed in descending order from the most highly correlated to the least highly correlated pair in the bipartite matching.

$$\text{CD}(\text{Net1}, \text{Net2}) = \sum_i \rho_{i,\text{Bipartite}(i)} - \rho_{i,\text{Semi}(i)} \tag{7}$$

We sampled ten networks with survival time ~191 (from the conditional CartPoleGen trained on the 151-200 survival time group) and ten networks with survival time ~29 (from the conditional CartPoleGen trained on the 0-50 survival time group) to represent good and bad networks respectively. Randomly selecting actions results in a survival time of 22, so 29 represents a bad network that is nonetheless acting better than random. The average all-pairs convergence distance in the good group and in the bad group are then computed, with the results summarized in Table 2. We visualize

the convergence distances in the hidden and output layer between selected pairs of CartPoleNets in Figures 5 and 6 respectively.

Table 2: Convergence of Good vs. Bad Networks

Group	Survival Time	Mean, Std CD (Hidden)	Mean, Std CD (Output)
Good	191	2.75, 1.96	**0.32**, 0.49
Bad	29	**3.13**, 1.7	0.09, 0.11

Higher CDs correspond to divergence, while lower CDs correspond to convergence.

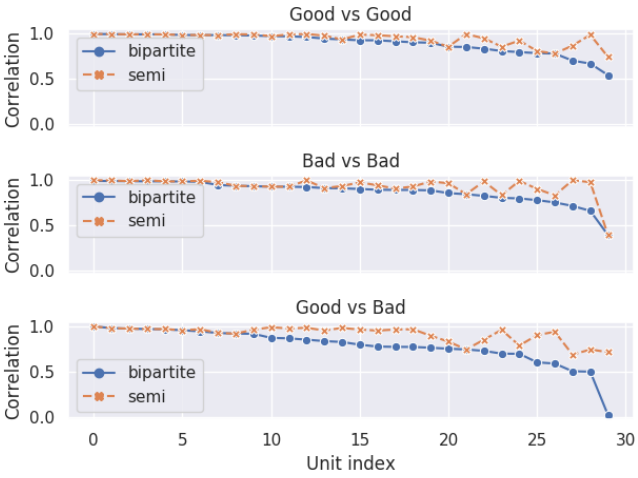

Figure 5: The figure shows correlations between *hidden* activations of a pair of good CartPoleNets, a pair of bad CartPoleNets, and a pair with one good and one bad CartPoleNet. For the networks used in this figure, the convergence distances between the pairs are 1.51, 1.75 **and** 3.91 **respectively.**

The data suggests that for the task of Cart-Pole that there are more ways to be successful than to be bad. In other words, given a random state in the environment, the good networks can diverge in their decision to move left or right to balance the pole, but the bad networks uniformly make the wrong decision. Surprisingly also, despite the good networks displaying divergence in their actions, they pick up on more convergent (good) representations.

It is quite interesting that there are more ways to balance a pole successfully than poorly, but the skills needed for the different paths to success are similar. We hypothesize that this is because the order of actions might be less important than the overall composition of the two actions. Consider a sequence of four actions. *{Left, Right, Left, Right}* would be highly negatively correlated with *{Right, Left, Right, Left}* but on average, they might produce the same outcome of keeping the pole balanced. On the other hand, *{Left, Left, Left, Left}* is highly correlated with *{Left, Left, Left, Left}* and they both cause the pole to quickly lose its balance.

Figure 6: The figure shows correlations between *output* activations of a pair of good CartPoleNets, a pair of bad CartPoleNets, and a pair with one good and one bad CartPoleNet. For the networks used in this figure, the convergence distances between the pairs are 0.32, 0.07 **and** 0.28 **respectively.**

4.2 Exploring the Latent Space

The latent space in CartPoleGen gives us semantic information about the kinds of networks that can be generated. We selected pairs of agent embeddings and sampled 20 new embeddings from $\alpha = 0.0$ to $\alpha = 1.5$ where α represents the coefficient of linear interpolation between the pair of embeddings. $0 < \alpha < 1$ represents interpolation, while $\alpha > 1$ represents extrapolation. The results are summarized in Figure 7.

The top left graph represents a pair of agent embeddings with a hidden CD of 1.82, the top right 12.5, the bottom left 2.13, and the bottom right 2.77. We observe that linearly interpolating within the latent space of CartPoleGen is not the same as simply interpolating within the weight space of CartPoleNet, given that CartPoleGen is non-linear in nature. In many cases, moving from a worse agent embedding to a better one tracks a similar improvement in survival time, as is the case in the top left and bottom right graphs. Furthermore, extrapolation results in a performance boost, up to a point.

However, we also observed many cases where interpolation resulted in agent embeddings whose network performed far worse or far better than the two embeddings used as endpoints for the interpolation. Interestingly, when the interpolated embeddings performed far better, it is often the case that the hidden CDs of the networks used for the two endpoint embeddings is fairly large. In the case of the top right graph, the hidden CD is in fact a few standard deviations above the mean.

4.3 Repairing Missing Weights

The generative model can be used to repair CartPoleNets with missing weights. We propose a simple rejection sampling based method (see Algorithm 1) to continuously sample new CartPoleNets

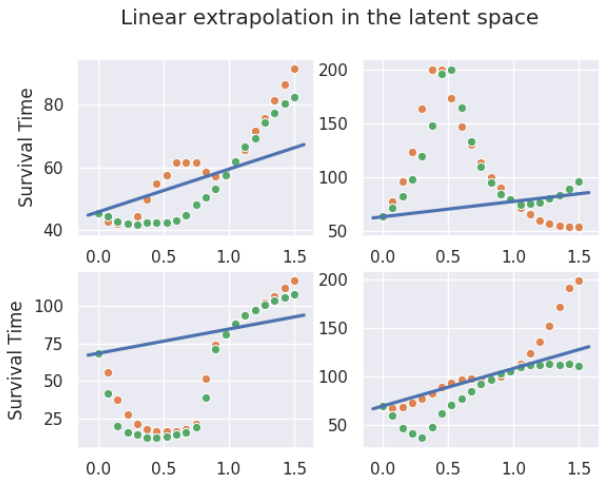

Figure 7: The x axis represents the coefficient of interpolation α, while the y axis represents the survival time of the sampled networks. The orange dots represent networks sampled from interpolating within the latent space, while the green dots represent networks interpolated within the weight space with the same coefficient of interpolation. The blue line is a straight line drawn from the survival time of the network sampled from the first agent embedding to the survival time of the network sampled from the second agent embedding.

from the model until suitable candidates are found to fill out the missing weights. We experiment with two possible criteria that can be used to pick the candidate.

$$W = \text{Existing} \cup \text{Missing} \tag{8}$$

$$C = \text{Candidate} \tag{9}$$

The *Missing Criterion* (see Equation 10) picks out the candidate who is most similar to the damaged CartPoleNet when we are only comparing the existing weights.

$$C^* = \arg\min_{C} \sum_{i \in \text{Existing}} (W_i - C_i)^2 \tag{10}$$

The *Whole Criterion* (see Equation 11) picks out the candidate who is most similar to the damaged CartPoleNet. This biases the selection towards finding candidates with tiny weights in the missing space.

$$C^* = \arg\min_{C} \sum_{i \in W} (W_i - C_i)^2 \tag{11}$$

We can probe the limits of our generative model for the task of weight repair by determining how much degradation can be reversed with a fixed computational budget (i.e. γ and k are fixed). To investigate this, we fix a given CartPoleNet, degrade it at a fixed level (i.e. zero out a fixed fraction of the weights at random), and repair it using the rejection sampling based algorithm proposed. The results are summarized in Figure 8.

Algorithm 1: Rejection sampling based method to repair missing weights in a CartPoleNet W. Let $ST()$ represent the survival time of a network.

$\gamma = 200$
$k = 10$
$\varepsilon = 5$
Sample γ networks from CartPoleGen
Pick k best candidates C^* using a *Criterion*
for $i \in [k]$ **do**
  if $| ST(C_i) - ST(W) | < \varepsilon$ **then**
    return Success, C_i
  end
end
return Failure, $\varnothing$

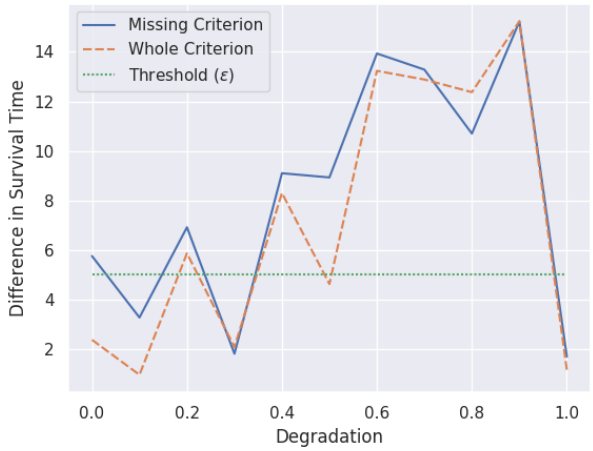

Figure 8: The figure shows the performance of the two criteria (in terms of the difference in survival time between the original network and the recovered network) used to repair missing weights at ten different levels of degradation. The threshold ε represents what we consider a successful level of recovery, so all the points below the threshold represent successful reversal of degradation.

We observe that the two criteria seem to perform similarly, with *Whole Criterion* performing slightly better, and we managed to successfully recover the network at some levels of degradation. While we do not recover the network completely (below the acceptable threshold of 5) in many cases, it is hopeful to note that there is partial recovery (the difference in survival times is at most 15). It is also interesting that it is possible to recover the network at complete degradation; this suggests perhaps that CartPoleGen has memorized this network.

The scheme described here can also be straightforwardly applied to the task of repairing (or verifying) corrupted weights instead of missing weights. We note that rejection sampling is an inefficient method of doing weight repair, and more sophisticated methods of conditional generation should be used if efficiency is of concern.

5 LIMITATIONS OF SUPERVISED GENERATION

We note three main limitations of the *Supervised Generation* method in learning agent embeddings.

5.1 High Sample Complexity

One of the primary drawbacks of the *Supervised Generation* method is the two-step process needed to first collect the data then train a generative model on it. This requires training a very large number of networks to provide the generative model with data. Figure 9 shows progressively worse approximations when we decrease the number of sampled networks by an order of magnitude.

In principle, an agent embedding does not have to be learned in this manner. For example, it might be possible to do *Online Generation* where a generative model learns to generate new networks on-the-fly with an online algorithm. *Online Generation* will probably be more sample efficient.

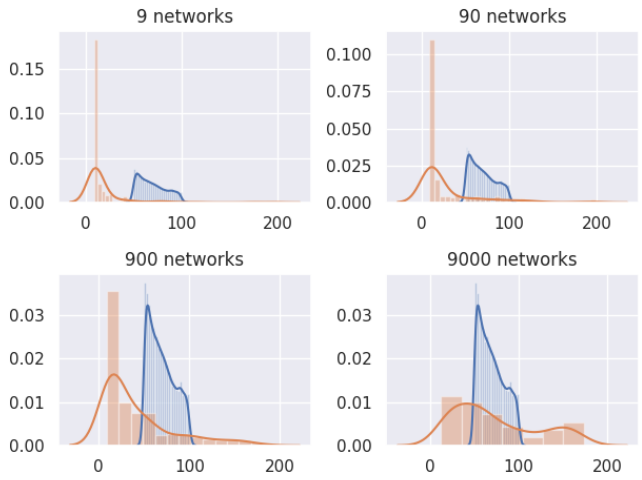

Figure 9: If we try to train the distribution in the 51-100 survival time group referred to in Figure 4 with fewer number of samples, we get worse approximations.

5.2 Subpar Model Performance

CartPoleGen does not approximate the training distribution very well (see Figure 4). This might potentially be fixed with a better generative model that also has access to online training data. For example, Bayesian HyperNetworks [16] might be a promising candidate.

5.3 Scaling Issues

We tried using a variational autoencoder to learn a 21840-dimensional weight vector for a small neural network that does MNIST image classification. Reinforcement learning agents that process images with CNNs would most likely contain weights at this order of magnitude at minimum. We trained it on a dataset of 10000 networks each with >95% accuracy, but none of the sampled networks managed to perform with >30% accuracy on a test set.

It might be difficult to scale the *Supervised Generation* method to large networks, even with significant advances made in generative modeling techniques. This is because even state of the art supervised generative models typically deal with data of much lower dimensions (<1000). A notable exception is WaveNet [33], but it deals with audio data which is relatively smooth and can tolerate high amounts of error, while the weights of a neural network are very discontinuous and are not robust to small amounts of additive noise.

6 POTENTIAL APPLICATIONS FOR AI

The ultimate challenge for neural network based generative systems is not generating images, sounds, or videos. The ultimate challenge is the generation of other neural networks. Learning agent embeddings is therefore a very difficult goal to accomplish, but we outline several potential applications for AI in general.

- AI systems powered by neural networks are often criticized for being uninterpretable. Agent embeddings provide us with a tool to gain insight into its internal workings and the space of possible solutions, which we have demonstrated with the task of pole balancing in this paper.
- The generative model can be conditioned to prevent it from generating networks that have undesirable properties like biases or security vulnerabilities. This is helpful for improving the fairness and security of AI systems. We showed how CartPoleGen can be used to repair weights in a network for example, which increases the data integrity of the system.
- It is helpful for an AI system to be able to generate worker AIs in a modular fashion. Each worker AI can be represented with its own agent embedding, and the generative model can be a factory that delivers a custom solution conditioned on the task given.
- Reinforcement learning agents perform better when they have access to a model of their environment. We think they will also perform better in multi-agent systems when they have access to compressed embeddings of other agents.

7 CONCLUSION

In this paper, we presented the concept of agent embeddings, a way to reduce a reinforcement learning agent into a small, meaningful vector representation. As a proof of concept, we trained an autoencoder neural network CartPoleGen on a large number of policy gradient neural networks collected to solve the pole-balancing task Cart-Pole. We showcased three interesting experimental findings with CartPoleGen and described the challenges of the *Supervised Generation* method.

8 ACKNOWLEDGMENTS

This research was supported in part by the US Defense Advanced Research Project Agency (DARPA) *Lifelong Learning Machines* Program, grant HR0011-18-2-0020. We would like to thank Peter Duraliev for his helpful suggestions in editing the paper.

REFERENCES

[1] Namrata Anand and Possu Huang. 2018. Generative Modeling for Protein Structures. (2018). https://openreview.net/forum?id=HJFXnYJvG

[2] Daniel Ashlock and Colin Lee. Agent-Case Embeddings for the Analysis of Evolved Systems. (????).

[3] Andrew G Barto, Richard S Sutton, and Charles W Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* SMC-13, 5 (1983), 834–846.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[5] Christopher M Bishop. 1997. Bayesian neural networks. *Journal of the Brazilian Computer Society* 4, 1 (1997).

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, and others. 2017. Interpretability of deep learning models: a survey of results. In *IEEE Smart World Congress 2017 Workshop: DAIS.*

[8] Oscar Chang and Hod Lipson. 2018. Neural Network Quine. *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)* (2018), 234–241. DOI: http://dx.doi.org/10.1162/isal_a_00049

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[10] Christopher Dann, Lihong Li, Wei Wei, and Emma Brunskill. 2018. Policy Certificates: Towards Accountable Reinforcement Learning. *arXiv preprint arXiv:1811.03056* (2018).

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[12] David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[13] John E Hopcroft and Richard M Karp. 1973. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing* 2, 4 (1973), 225–231.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[16] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. 2017. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759* (2017).

[17] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*. 3675–3683.

[18] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv preprint arXiv:1804.08838* (2018).

[19] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. 2015. Convergent Learning: Do different neural networks learn the same representations?. In *Feature Extraction: Modern Questions and Challenges*. 196–212.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055* (2018).

[21] Christos Louizos and Max Welling. 2017. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961* (2017).

[22] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 746–751.

[23] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1602.01783* (2016).

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[26] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *arXiv preprint arXiv:1802.03268* (2018).

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

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and others. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[29] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[30] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and others. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[31] Sean C Smithson, Guang Yang, Warren J Gross, and Brett H Meyer. 2016. Neural networks designing neural networks: multi-objective hyper-parameter optimization. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 1–8.

[32] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*. 3483–3491.

[33] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[34] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. 2018. Programmatically Interpretable Reinforcement Learning. *arXiv preprint arXiv:1804.02477* (2018).

[35] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. 2016. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*. 613–621.

[36] Taihong Xiao, Jiapeng Hong, and Jinwen Ma. 2017. DNA-GAN: Learning Disentangled Representations from Multi-Attribute Images. *arXiv preprint arXiv:1711.05415* (2017).

[37] Yantian Zha, Yikang Li, Sriram Gopalakrishnan, Baoxin Li, and Subbarao Kambhampati. 2018. Recognizing Plans by Learning Embeddings from Observed Action Distributions. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2153–2155.

[38] Quanshi Zhang and Song-Chun Zhu. 2018. Visual Interpretability for Deep Learning: a Survey. *arXiv preprint arXiv:1802.00614* (2018).

[39] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).