ON THE APPLICATION OF T-SNE AND SVCCA FOR VISUALIZING AND MEASURING NEURAL NETWORK REPRESENTATIONS IN REINFORCEMENT LEARNING

Cody Rosevear
Department of Computing Science
University Of Alberta
Edmonton, AB T6G 2E8, Canada
rosevear@ualberta.ca

ABSTRACT

Artificial intelligence has seen a number of high profile achievements in recent years, largely due to innovations in deep learning and the concomitant growth and increased ease of access to computing power and data.

Despite these improvements to the state-of-the-art, the inner workings of neural networks remain theoretically murky, and there is no broad consensus concerning how best to interpret the representations that they learn.

Recent work has proposed a number of explicit formalisms that seek to render network representations more amenable to analysis and comparison.

SVCCA (Raghu et al., 2017) in particular offers a flexible and computationally cheap similarity measure for representations. However, it has yet to be applied within a reinforcement learning context, and its utility assessed thereof.

The following work assesses the degree to which SVCCA, as applied in a few simple reinforcement learning environments, coheres with other measures of a neural network, such as its online/offline performance, its learned value functions, convergence rates, and dimensionality reduction techniques such as t-distributed stochastic neighbor embedding, given different layer widths and initialization schemes for the network.

Results indicate a fair degree of conceptual coherence between the value functions, tsne representations, and SVCCA similarity scores as computed for the final hidden layer of a neural network. Correlations between offline performance and the evolution of the representations to their final state, and the impact of network width on the rate of convergence to the final representation, as measured by SVCCA, are inconclusive, and in need of further analysis.

1 INTRODUCTION AND RELATED WORK

The field of artificial intelligence (AI) has seen rapid progress in recent years, due in large part to the steady increase of data and cheap computing power afforded by Moore's Law (Moore, 1998). This trend towards ever more efficient computation and storage capability has had a significant impact on the practicality of machine learning methodologies such as deep learning (Goodfellow et al., 2016) and reinforcement learning (RL) (Sutton & Barto, 2018), which were previously thought to be too computationally burdensome for many tasks, but have since been shown to be practical with respect to a number of applications, such as image (Krizhevsky et al., 2012) and speech recognition (Graves et al., 2013), multi-core processer managment (Martinez & Ipek, 2009), and autonomous helicopter flight (Abbeel et al., 2007). Moreover, research actively combining these two methodologies together, alongside innovations in heuristic search (Chaslot et al., 2008), have yielded a number of high performing algorithms in such difficult problem domains as heads-up no-limit Texas hold em (Moravčík et al., 2017), Atari style video games, (Mnih et al., 2013), and the ancient board games of Go (Silver et al., 2017), and Shogi (Silver et al., 2018).

However, despite these successes there remains a lack of a broad theoretical consensus on how best to interpret the representations that are learned by neural networks in such a way so as to render the decisions such systems make intelligible to human users and practitioners. This is problematic not only insofar as a lack of understanding of how neural networks achieve the results that they do necessarily limits one's ability to improve upon current learning algorithms, but prior work has demonstrated that the representations learned by neural networks within both the tradional deep learning context (Goodfellow et al., 2015), and deep reinforcement learning (Huang et al., 2017; Behzadan & Munir, 2017), are prone to overfitting in subtle ways that can only be exposed when the networks are subjected to specific adversarial examples.

As the state of the art continues to advance and the research community seeks to harness the benefits of practices such as transfer learning (Taylor & Stone, 2009) and auxiliary task based learning (Jaderberg et al., 2017; Cabi et al., 2017) the representations learned by neural networks will invariably get even more complicated. As such, it is crucial that reliable methods for interpreting neural network representations be developed, to assist in creating new learning methods and training procedures that avoid the learning of brittle representations that fail in critical contexts, and instead reliably produce representations that admit of good performance across a wide range of tasks: one of the key milestones to achieving general purpose AI.

Prior work (Bengio et al., 2013) has attempted to characterize what constitutes a 'good' representation, and in what way the nature of the representation impacts the performance of learning agents. The general idea is that representations serve as a means by which very general priors concerning the world can be encoded within a learning system, based on global properties of the representations, such as whether or not they are sparse, distributed, hierarchical, and capable of disentagling the multiple factors of variation that account for the diversity of outputs in a given data generation process.

While such a characterization of the quality of representations is fruitful insofar as it accounts conceptually for how representational quality may impact the end-quality of a machine learning model, many of these properties are fairly high-level, in the sense that how to go about guaranteeing their presence within any given machine learning model instance, while simultaneously maintaining good performance on the task at hand, is underspecified as regards the details of the training methodology, architecture, and so forth, that one should use. Such details are still an active area of research, and, indeed, are often implicit assumptions of any given representation learning paradigm that one chooses to employ (e.g. artificial neural networks, probabilistic graphical modeling, manifold learning, etc...).

In order to move beyond making general paradigmatic assumptions when constructing machine learning models, to the point where analyzing and iterating on the details of the specific representations themselves becomes a plausible part of the training process, an explicit formalism for what composes a representation instance is required. Moreover, this formalism must allow for some definition of similarity between representations that allows for one to compare between different representations in meaningful ways.

Recent work has tackled this more particular form of the problem on a number of fronts: Li et al. (2015) define a measure of similarity between layers in a network in terms of the correlation or mutual information of the mean and standard deviation of neuronal activations over a dataset, and ask how many permutations are required in order to bring layers of different networks into alignment, with respect to the measure used (correlation or mutual information). This is a simple, intuitive first approach to considering the question of how to measure a representation, but is limited to networks that share the same architecture.

Raghu et al. (2017) define neuronal representations in terms of activation vectors over the entire data set, and layers in terms of spanning sets of these activation vectors, to yield a subspace model of network representations, which allows for the direct comparison of networks with different architectures via a new statistical method (See section 2.7) proposed in the same work.

Wang et al. (2018) build upon the work of Raghu et al. (2017), by extending the subspace model of network layer representations to include the notion of a 'match' between given neurons, that can be decomposed into simpler matches, where the number of such simple matches is used as a measure of similarity at the neuronal level, and neuronal matches are treated as a measure of overall similarity between network layers. While conceptually quite intuitive, accompanying experimental work re-

vealed a suprising lack of similarity between layers of the same convolutional network trained from different initializations, suggesting that the utility of the measure is not as of yet clear cut.

This highlights the importance of evaluating measures of representational similarity in simple contexts, with clear expectations as to how the results will turn out, and a relatively clear path, on account of the environment's simplicity, as to how to proceed and re-assess the method if they do not. In that spirit, the following work seeks to evaluate the merit of SVCCA as a measure of representational similarity within the context of RL, in a number of simple gridworld environments, to explore to what extent variations in initializaton schemes and network architecture alter the learning dynamics as measured by SVCCA, and observe whether these variations match prior intuitions (See section 3) with respect to how they should cohere with other relevant measures of the network, such as online/offline performance, the learned value functions, and visualizaton techiques, such as t-distributed stochastic neighbour embedding, that have a prior history of application within deep RL (Mnih et al., 2013).

2 BACKGROUND

2.1 MARKOV DECISION PROCESSES

A traditional reinforcement learning problem as described in Sutton & Barto (2018) is composed of two parts: an agent, and an environment, formulated as a markov decision process (MDP). An MDP defines a set of states $\mathcal{S}$, and actions $\mathcal{A}$, wherein for each time step $t = 0, 1, 2...n$, where $n \in \mathcal{N}$, the agent inhabits a state $s \in \mathcal{S}$, and chooses an action $a \in \mathcal{A}$. Each such state action pair (s_t, a_t) yields a corresponding scalar reward, $r_t \in \mathcal{R}$, while transitioning to a new state s_{t+1} according to the environment transition function $p(s_{t+1}, r_t|s_t, a_t)$, which specifies the probability of ending up in state s_{t+1} and achieving reward r_t given the current state s_t and action a_t.

The transition function is assumed to satisfy the Markov Property, which states that the conditional probability of the next state is dependent only on the previous state-action pair, such that $p(s_{t+1}, r_t|s_t, a_t) = p(s_{t+1}, r_t|s_t, a_t, s_{t-1}, a_{t-1}...s_{t-k}, a_{t-k}), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}\ \forall t, k\ \in \mathcal{N}$, such that $k < t$. In practice this assumption will not always hold, but many RL algorithms are robust enough that so long as the violation is not too exteme they will nevertheless perform well enough for the task at hand.

Once the agent transitions to a new state upon completing an action at time t, the agent then takes a new action a_{t+1}, yielding a new reward r_{t+1} and state s_{t+2}, and the cycle continues, either until termination in the case of an episodic task with terminal states, or indefinitely, in the case of a continuing task. See figure 1 for a graphical depiction of the agent-environment interaction loop.

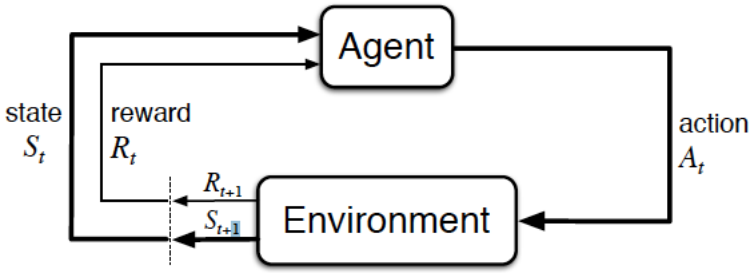

Figure 1: The reinforcement learning agent-environment loop. Source: (Sutton & Barto, 2018)

2.2 POLICIES AND VALUE FUNCTIONS

The agent chooses actions according to a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which defines the probability of selecting action a given state s. The return, denoted G_t, is defined as the sum of the discounted rewards since time t. That is, $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$, where $0 \leq \gamma \leq 1$ is the discount factor, k is the index of the number of time steps into the future, R_k is the reward at time step k, and T is the final time step. T may be a natural number for episodic tasks with terminal states, or, in the case of continuing tasks, be assigned the value of ∞, in which case it must be that $\gamma < 1$, to ensure that no returns diverge. The discount factor is used to specify how far-sighted the agent is, where 0 indicates

a myopic agent concerned with maximizing the immediate reward and 1 specifies an agent that takes into account the full value of future states when deciding its next action based on its current policy.

The state value function $V_\pi(s)$ is defined with respect to a particular policy π, such that it is equal to the expected return if one were to follow policy π starting in state s. More formally, $V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_t + \gamma G_{t+1}|S_t = s]$. Crucially, since state values are defined in terms of expected return, which itself can be defined in terms of the current reward and future returns, the state value function can also be defined recursively in terms of future state values, weighted by their probability of occurring according to both the environment dynamics and the current policy probabilites for actions. This is the celebrated Bellman Equation for state values (Bellman, 1957), and is denoted by equation 1. The related Bellman Equation for state-action values, which denotes the value of inhabiting state s, taking action a, and following policy π thereafter until termination, can likewise be recursively defined, as in equation 2.

$$V_\pi(s) = \sum_a p(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma V_\pi(s')] \tag{1}$$

$$Q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \sum_{a'} \pi(a'|s')\gamma Q_\pi(s', a')] \tag{2}$$

For any given MDP there exists at least one optimal policy π_* such that for all policies $\pi, V_{\pi*}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$. These optimal policies can be found by acting greedily with respect to the corresponding optimal state-action value function, or performing a one-step look-ahead on the optimal state value function, denoted by equations 3 and 4, respectively.

$$Q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} Q_*(s', a')] \tag{3}$$

$$V_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_*(s')] \tag{4}$$

Assuming the presence of a complete model of the environment transition function, and suitable computational resources, the various value functions with respect to a policy π can be approximated via dynamic programming (Bellman, 1957), using the corresponding Bellman Equation for a given value function as an update rule to compute it. Subsequently, once the value function has been suitably approximated with respect to the current policy π, one can improve upon policy π by constructing a new policy, π', that behaves in a deterministically greedy fashion with respect to the currently approximated value functions by setting $\pi'(s) = \arg\max_a Q(S, a) \forall s \in \mathcal{S}$, which is guaranteed by the policy improvement theorem (Bellman, 1957) to yield a superior policy such that $V_{\pi'}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$.

2.3 Q-LEARNING

When a problem does not admit of a complete model of its environment dynamics, but the state and action space of a problem is still tractably finite, one can store learned estimates of all state-action values within a table, and simply lookup a given value in order to use it, or modify it directly in light of new exeprience. Tabular Q-learning (Watkins, 1989) does exactly this, approximating and storing the optimal state-action value function Q(s, a) directly. It does so by learning off policy: allowing a separate behaviour policy π to select actions and generate a trajectory of experience tuples (s_t, a_t, r_t, s_{t+1}) from which estimates of the optimal state-action values can be computed, according to equation 5, where α is a positive scalar specifying the learning rate and γ is the discount factor. In essence, the learning algorithm seeks to shift the estimate by a multiple of the so called TD-Error δ at time t, where $\delta_t = R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$. This shift serves to push the current estimate closer to the current target value for the state: $R_t + \gamma \max_a Q(s_{t+1}, a)$.

$$Q_(s_t, a_t) = Q_(s_t, a_t) + \alpha[R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{5}$$

2.4 FUNCTION APPROXIMATION

Oftentimes, however, the number of states is combinatorially explosive, and it is not possible to store the state-action values in a table directly, nor possible to visit all relevant states so as to learn about them in a reasonable amount of time. What is required, therefore, is the capacity for the agent to generalize what it learns about those states it does encounter to suitably similar, but unencountered, states.

Parametric function approximation is one such way of addressing the need for generalization. In this case, the value of a state-action pair is represented as the output of a function, paramterized by a set of weights $\theta_t \in R^{n \times m}$, which takes as input a state vector $x(s_t) \in \mathbb{R}^k$ and computes a scalar $r \in \mathbb{R}$ representing the value of the current state (or state-action pair). Like in the tabular case, the error between the target value and the current prediction should be reduced in accordance with a specified learning rate, only in this case the shift occurs indirectly by updating the value of the function parameters at the current time step θ_t. Typically these weights are updated according to some variant of stochastic gradient descent (SGD) (Robbins & Munro, 1951), via a gradient update rule whose form is dependent on the specifc functional form of the approximator.

2.5 ARTIFICAL NEURAL NETWORKS

Artificial neural networks (ANN) are one method of function approximation that are particularly powerful, with the capacity to represent virtually all functions of practical interest (Cybenko, 1989) assuming the satisfaction of a few unrestrivctive conditions. An ANN is composed of an input layer, some number of hidden layers, and an output layer. Each layer is composed of a number of neuronal units, each of which is connected to every neuronal unit in the preceding layer by a real valued scalar weight. Each neuron takes in the sum of the outputs of the neurons in the previous layers to which it is connected, multiplied by the corresponding weight for that connection. The result of this summation is then passed through an activation function to produce the output for that neuron, which then propagates its own output further down the network, continuing the process until the output layer is reached and the network emits its final values. See Figure 2 for a graphical depiction of a simple feedforward artificial neural network.

In order to learn to approximate a desired function, example inputs are provided to the first layer of the network. A loss between the predicted output and the target is computed, and the weights of the various layers are adjusted via the backpropagation algorithm (Werbos, 1974). The goal is to iteratively update the weights to traverse the space of the cost function and find a set of parameters that minimize the loss and predict the appropriate value (or close approximations thereof) for new inputs.

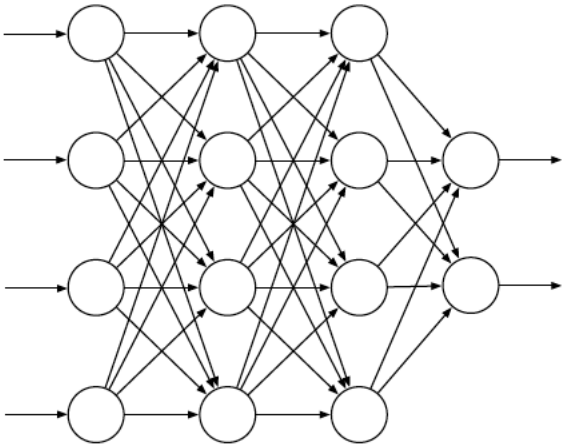

Figure 2: A simple feedforward artificial neural network. Source: (Sutton & Barto, 2018)

2.6 T-DISTRIBUTED STOCHASTIC NEIGHBOUR EMBEDDING: T-SNE

Hinton & Roweis (2003) introduced stochastic neighbour embedding (SNE) as a dimensionality reduction and data visualization technique. It converts the Euclidean distance between points in a data set into conditional probabilities that represent the similarity of points to one another. Concretely, the similarity of point x to point y is the conditional probability, $p(x|y)$, that a Gaussian distribution with mean x would yield y if data points were selected at random in proportion to their density in the distribution centered at x. A corresponding similarity measure is defined for the low dimensional analogs of the high dimensional data points, and a gradient descent algorithm, with momentum (Qian, 1999), is used to minimize the sum of the Kullback-Leibler divergence (Kullback & Leibler, 1951) between the 2 probability distributions, over all data points, in an effort to find the best representation of the high dimensional data in the low dimensional space. The variance of the high-dimensional Gaussian used for the similarity measure is automatically determined by the algorithm based on the user chosen perplexity parameter $Perp(P_i) = 2^{H(P_i)}$, where $H_(P_i)$ is the Shannon entropy (Shannon, 1948) and P_i denotes the conditional probability distribution over all other datapoints given data point i. The algorithm will find the relevant variance for any selected perplexity value.

t-distributed SNE (Maaten & Hinton, 2008) maintains the same formulation of similarity in terms of probability distributions, but alters the cost function in 2 ways: it alters the probabilities used to define the cost function (in terms of KL) so that they exhibit symmetry, which has the effect of yielding a cost funtion with a simpler gradient that allows for easier optimization, and it replaces the Gaussian distribution used to determine the probabilities of the points in the low dimensional space with a student's t-distribution, to address the 'crowding problem' (Maaten & Hinton, 2008), whereby moderate and large distances in the high-dimensional space cannot be faithfully represented in lower dimensional spaces due to the larger amount of space afforded by more dimensions, resulting in long and even moderate distance data points crowding together in the lower dimensional space in an uninformative way.

2.7 SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS: SVCCA

Singular Vector Canoncial Correlation Analysis (SVCCA) (Raghu et al., 2017) defines the representation of a neuron as a vector $z \in \mathcal{R}^n$ composed of all of the activations for that neuron over a given data set of size n. A given layer l is therefore defined as the subspace of $\mathcal{R}^n$ spanned by these neuronal vectors.

SVCCA takes in as input two sets of such vectors, l_1, l_2 each representing a given layer of a network, and performs a singular value decomposition (SVD) upon the sets, to create sub subspaces $l'_1 \subset l_1$, $l'_2 \subset l_2$ which account for 99% of the variance of the original vectors. Concretely, given the resultant set of singular vectors $m_1...m_k$, and singular values $\lambda_1..\lambda_k$, after applying SVD, the vectors are pruned so that only the top k' vectors are retained, where k' is the smallest value such that $\sum_{i=1}^{k'} |\lambda_i| >= 0.99 \sum_{i=1}^{k} |\lambda_i|$.

These remaining singular vectors are then subject to Canonical Correlation Analysis, a statistical technique which seeks to find linear transformations that maximally correlate the subspaces spanned by l'_1 and l'_2. The algorithm then yields as output a pair of spanning sets for the transformed subspaces, as well as the corresponding correlations between the spanning elements of the transformed subspaces. These correlations are then used to compute the SVCCA Similarity of the two layers, $\bar{\rho}$, defined by equation 6

$$\bar{\rho} = \frac{1}{min(l'_1, l'_2)} \sum_i \rho_i \tag{6}$$

where $min(l'_1, l'_2)$ is the cardinality of the smaller of of spanning vectors out of both of the layers, and ρ_i is the correlation between the corresponding vectors in each such spanning set.

3 QUESTIONS AND HYPOTHESES

There are a number of questions one can ask concerning the relationship a measure of representational similarity such as SVCCA might have to other facets of the neural network learning process.

3.1 THE RELATIONSHIP BETWEEN SELF-SIMILARITY AND PERFORMANCE

Is there some type of quantitative relationship, either of linear correlation (Galton, 1889; Pearson, 1895) or more general monotonicity (Spearman, 1904), between the pattern of convergence of a network with its offline performance curve, where performance is defined in terms of the number of steps per episode that the agent requires to reach its goal state? To the extent that good representations are supposed to improve performance, for those networks that end up performing well by the end of the run, it is reasonable to suspect that a strong negative correlation will exist between the degree of similarity to the final convergent representation and the performance of the algorithm.

3.2 THE SIMILARITY OF CONVERGENT REPRESENTATIONS

Do different network initializations and architecture widths converge to reasonably similar solutions within the same span of time? Do the wider networks converge to more similar solutions compared to narrower ones, or vice versa, across different initializations?

For a simple problem domain like the gridworld where there are a few (or possibly even one) optimal strategies, it is intuitively plausible that the final layers of networks with different initialization schemes, but comparable final performance, should converge to a similar representation.

To the extent that this is not true, this would be surprising, and worthy of further investigation, as it may point to a problem with the measure in question, given the simplicity of the environment. This is not necessarily the case with more complex environments, since they might admit of multiple ways to solve a problem, as suitably diverse optimal strategies in a more difficult environment could reasonably be expected to yield representations that are dissimilar in non-trivial ways; but one of the benefits of experimentation within a simple environment is that one's expectations regarding the representation can be made quite clear, even with respect to high performing agents. Relatedly, if poor performing networks are highly similar to good ones, this is also suggestive that something is amiss concerning the measure in use. It seems likely, therefore, that within the context of a simple gridworld experiment as presented here, to the degree that two different networks perform similarly, it is to be expected that their final layer convergent representation will also be similar.

3.3 NETWORK LAYER WIDTH AS A FACTOR IN THE RATE OF CONVERGENCE TO A REPRESENTATION

In addition to the dynamic nature of a network's representation and its final representation, there is also the question of the rate of convergence of a given network, particularly as this relates to the network size, in terms of either the number of layers or their widths. It is a plausible hypothesis that larger networks, taking longer to train to reach a high level of performance will, similarly, take longer in general to converge to their final representation. This makes sense insofar as a smaller number of parameters to tune in the case of smaller networks means that there are fewer degrees of freedom by which a network can differ from its final configuration; and, as such, the smaller parameter space should allow a simpler network to converge to a final representation more quickly than a larger counterpart, meaning that it will reach a high degree of self-similarity with its final layer more quickly than comparably wider networks.

3.4 SVCCA AND VISUALIZATION

Finally, SVCCA is a single number metric. This is valuable insofar as one is interested in quantifying similarity in a manner that allows for fairly quick and easy comparisons, and where a clear, objective anwer regarding the relationship between multiple networks is required (i.e., which of network X and Y is most similar to network Z). However, a single metric of necessity compresses a lot of information that might otherwise be useful in interpreting a network layer, such as might be learned by exploring a more visual representation of a given layer.

Parameter	Value
α (Learning Rate)	0.1
γ (Discount Factor)	0.99
Max ϵ (Starting Exploration Rate)	1
Min ϵ (Final Exploration Rate)	0.01

Table 1: Discrete Gridworld: Tabular Agent Hyperparameters

While SVCCA provides additional output in the form of the actual spanning sets of neurons for the maximally correlated subspaces, which might be harnessed for such visualization, comparing to what extent the simliarity values of SVCCA cohere with known, pre-existing techniques for visualization within a deep reinforcement learning context, such as value function heatmaps and t-sne, allow for an additional method of assesing SVCCA: to the degree that the results of SVCCA make sense in light of such visualizations, it is a type of assurance that SVCCA is not removing information that other techniques consider worth preserving.

4 Experiments

4.1 Discrete Gridworld

4.1.1 Environment

The discrete environment is a small maze composed of a six by nine set of fully observable states. The states can either be inhabited by the agent, or contain an immovable obstacle object that prevents the agent from occupying that space. If the agent attempts to move into such a space it simply remains within the space it currently inhabits. The agent can select from a set of actions that allow it to travel one block in either of the four compass directions: North, South, East, or West. The task is episodic: the agent starts in a specified start state at the beginning of each episode and must navigate to a specified goal state within 1000 time steps, otherwise the episode terminates by default. The start and goal states are static and do not change on a per episode basis, and the agent is penalized with a reward of -1 for each time step spent not in the goal state, which rewards the agent with a 0 when transitioned to. This is to ensure a rich reward that incentives the agent to reach the goal as quickly as possible. See Figure 3 for a graphical depiction of the discrete gridworld environment.

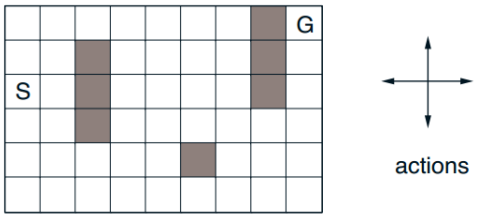

Figure 3: Discrete Gridworld Maze. Source: (Sutton & Barto, 2018)

4.1.2 Agents

Tabular Q-learning (Watkins, 1989) served as both a sanity check and baseline agent with respect to both online and offline evaluations. See table 1 for a listing of the hyperparameters as used by the tabular agent.

The neural network agents all used a two-layer fully connected feedforward architecture, with the same number of neurons in each layer, with 3 different variants for the number of neurons; namely, 5, 50, and 150. The output layer contained four nodes, one for each action given the current state as input. The hidden layers and output layer used relu and linear activation functions, respectively, and all neural agents used a variant of DQN (Mnih et al., 2013) which employed a target network whose weights were updated every 1000 time steps, but without experience replay or any form of gradient clipping. See Figure 5 for a listing of the hyperparameters as used by all of the neural network agents.

Parameter	Value
α (Learning Rate)	0.001
γ (Discount Factor)	0.99
Max ϵ (Starting Exploration Rate)	1
Min ϵ (Final Exploration Rate)	0.01
Linear Annealing Rate (Exploration Decay Rate)	0.05
Target Network Update Frequency	1000
Layer One Neurons	5, 50, 150
Layer Two Neurons	5, 50, 150

Table 2: Discrete Gridworld: Neural Network Hyperarameters

The weight initialization procedure was varied in order to test its impact on performance and representation learning. Variants tested include random uniform, glorot normal (Glorot & Bengio, 2010), and he normal (He et al., 2015) initialization.

All agents used one-hot encodings for the state vector and an epsilon-greedy exploration strategy, wherein they selected from among all possible actions at random with probability $\epsilon/|\mathcal{A}|$, and selected the action that maximized over the state-action values for the current state with probability $1 - \epsilon$. Exploration started off at 1.0 at the beginning of each run, and was annealed linearly at the end of each episode at a rate of 0.05 per episode until reaching the minimum value of 0.01. Neural agents used the Adam optimizer (Kinga & Ba, 2015) with a mean squared error loss, denoted by equation 7, where $\hat{y}_i$ is the target state-action value and y_i is the agent's current prediction for the state-action value, as returned by the target network.

$$\frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2 \tag{7}$$

4.2 Continuous Gridworld

4.2.1 Environment

The continuous gridworld environment is also an episodic task, where the agent must navigate from a start position to a goal position. The start and end states are static, and do not change between each episode. The agent starts at position (0.50, 0.50), behind a wall which it must navigate around in order to reach within 0.01 of position (1.0, 1.0), within 10000 time steps. The environment rewards the agent with -1 at each time step, except for the goal state, which provides a reward of 0. The agent can select an action movement at each time step to move an increment of 0.01 in one of the four compass directions: North, South, East, and West. A small noise term, selected uniformly at random from the range +=/ 0.005, is added to each action to induce a continuous state space. See Figure 4 for a graphical depiction of the environment, and Table 3 for a listing of the environment parameters.

Parameter	Value
Action Effect Size	0.01
Noise Factor Max	0.005
Noise Factor Min	-0.005
Noise Probability Distribution	Uniform
Goal Tolerance	0.01

Table 3: Continuous Gridworld Environment Parameters

4.2.2 Agents

Sarsa lambda with tile coding (Sutton, 2018) was used as a performance baseline and visualization reference point. See Table 4 for a listing of the hyperparamters as used by the agent.

The neural network agent utilized the same feedforward architecture as in the discrete gridworld case: two fully connected hidden layers with the same number of neurons per and 4 output nodes representing each action, wlthough only one variation of network width, of 150 neurons per layer,

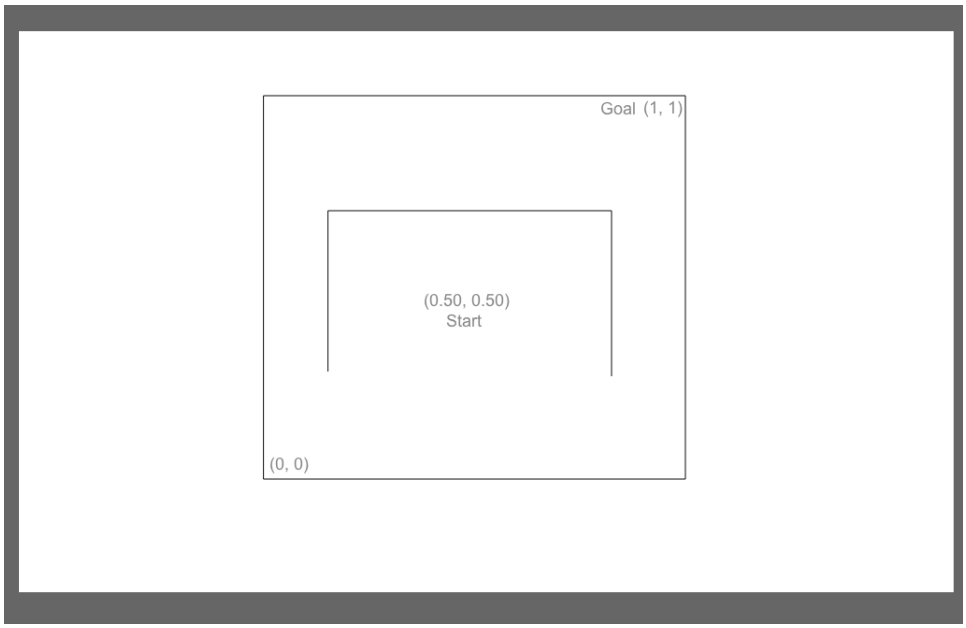

Figure 4: Continuous Gridworld. Source: The Author

Parameter	**Value**
α (Learning Rate)	0.15
γ (Discount Factor)	0.99
Max ϵ (Starting Exploration Rate)	1
Min ϵ (Final Exploration Rate)	0.1
Linear Annealing Rate (Exploration Decay Rate)	0.05
Lambda (Eligibility Trace)	0.90
Num Tiles	8

Table 4: Continuous Gridworld: Sarsa Lambda Hyper-parameters

was used. Both experience replay, with a buffer size of 10000, and corresponding mini-batch size of 10 were used with combined DQN (Zhang & Sutton, 2017), which composes each mini-batch out of a combination of the agent's current experience tuple in addition to 9 experience tuples selected uniformly at random from the replay buffer. Learning was delayed until the buffer is full, which occurs roughly after the first episode, due to a high exploration rate. As in the discrete case, actions are selected according to an epsilon-greedy style, with ϵ starting at 1 and linearly annealing over time at a rate of 0.05 per episode to a minimum of 0.1. All agents took in the x and y co-ordinate as input.

Parameter	**Value**
α (Learning Rate)	0.001
γ (Discount Factor)	0.99
Max ϵ (Starting Exploration Rate)	1
Min ϵ (Final Exploration Rate)	0.1
Linear Annealing Rate (Exploration Decay Rate)	0.05
Replay Buffer Size	10000
Target Network Update Frequency	1000
Layer One Neurons	150
Layer Two Neurons	150
Mini-Batch Size	10

Table 5: Continuous Gridworld: Neural Network Hyperarameters

4.3 IMPLEMENTATION

All experiment code implements the RL glue interface (Tanner & White, 2009). Neural networks were implemented using the Keras deep learning framework (Chollet et al., 2015) with Theano (Al-Rfou et al., 2016) serving as the backend.

SVCCA was computed using the numpy matrix (Oliphant, 2006) and Pandas (McKinney et al., 2010) data manipulation libraries, in conjunction with the code open sourced by Raghu et al. (2017). All correlation computations were handled by the open source computing library Scipy (Jones et al., 2001); t-sne computations were performed using the implementation provided by scikit-learn (Pedregosa et al., 2011), and all charts were produced with MatplotLib (Hunter, 2007).

Source code for all of the experiments can be found at: `https://github.com/Rosevear/Capstone`

5 RESULTS AND DISCUSSION

5.1 DISCRETE GRIDWORLD

For the discrete case, performance results were collected and averaged over 30 runs of 3500 episodes each, with all runs for each agent being initialized with the same random seed to guarantee both reproducibility and ensure identical elements of randomness in the agent's experiential trajectories, to allow for a fair comparison across varying agent architectures and intitialization schemes. Offline evaluation occurred every 50 time steps until episode termination, and consisted of halting all learning and random action selection for a trial episode and recording the number of time steps taken to reach the goal.

5.1.1 PERFORMANCE

Online performance on the discrete gridworld task across all agents and initializations performed quite well asymptotically, although all of the neural network results are far noiser than the tabular agent. Previous runs of the experiment where the minimal epsilon was set to 0.1 were less noisy, but tended to have poorer offline performance relative to the currently reported vlaue of 0.01. Somewhat surprisingly, although all initialization schemes and architectures converged to a similar level of online performance, for both the 5 and 150 neuronal architectures, the random initialization scheme seemed to perform better earlier on in comparison to the he and glorot intializations with respect to the former network, and when compared to the glorot initialization, with respect to the latter. The 50 neuron network runs were more similar across all initializations than the other network widths.

This propensity for poorer earlier performance could be accountd for by the fact that the he and glorot initilization schemes are more concerned with avoiding vanishing/exploding gradients in deeper neural networks, and so the benefits of the initialization are less salient in shallower networks of 1 or 2 layers, such that random initialization is actually preferrable because it does not involve any trade offs that might be being made by the other initialization schemes to protect against the issues encountered in deeprer networks. These results are, however, merely suggestive, and in need of an analysis of statistical significance.

The offline performance across networks displayed a similar pattern, in line with the online performance. See figures 5 to 8 for a graphical depiction of the results for both the online and offline cases.

5.1.2 VISUALIZING LEARNED REPRESENTATIONS

The value function map plots the negative of the maximum state-action vaue for a given state, to represent the expected cost from that state to the goal state.

The shape of the value function plot varied considerably depending on the network width used, but remained fairly similar across different initialization schemes. With respect to the latter, in the 5 neuron wide network case, the random initialization performed best (excluding the tabular case), and its value functions most closely resembled that of the tabular case, as compared to the other initialiation schemes. However, even the 50 and 150 wide networks managed to perform quite well

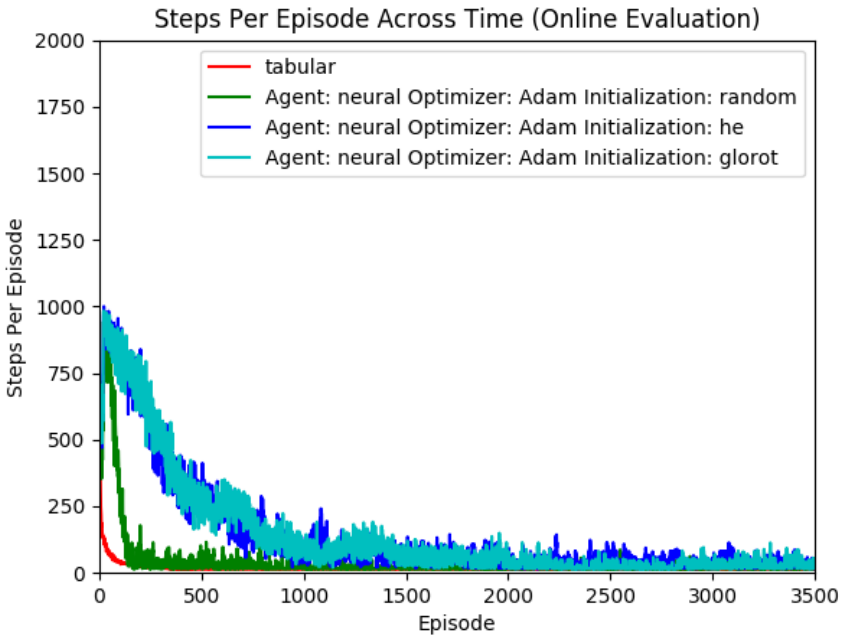

Figure 5: Online Performance 5 Neuron Architecture

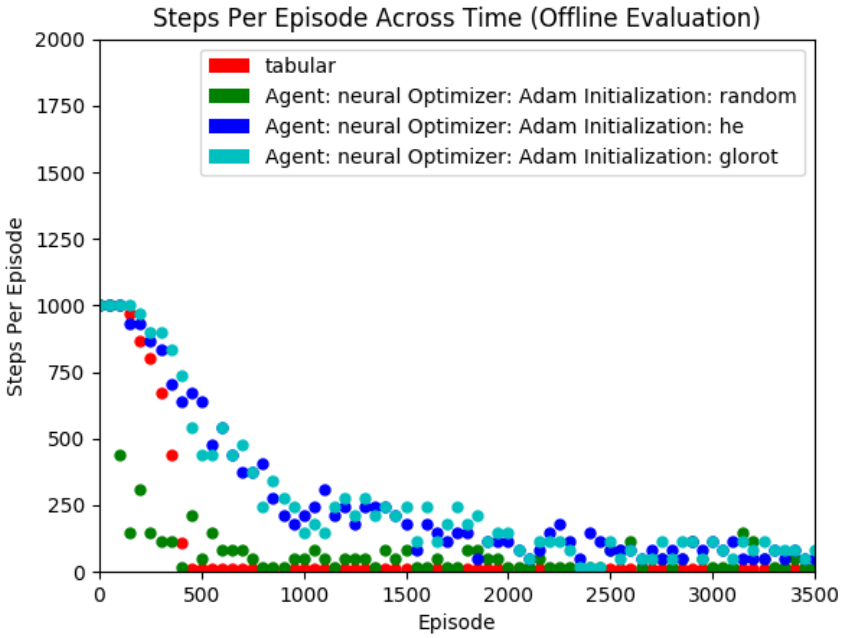

Figure 6: Offline Performance 5 Neuron Architecture

in the long run, but their value functions are much less similar to the tabular case, when compared to the 5 neuron wide architecture. In general, they display an overall shape and topology such that low value states and high value states are positioned relative to each other in a similar way to the tabular and narrow network value functions, but the absolute values are different. This suggests that

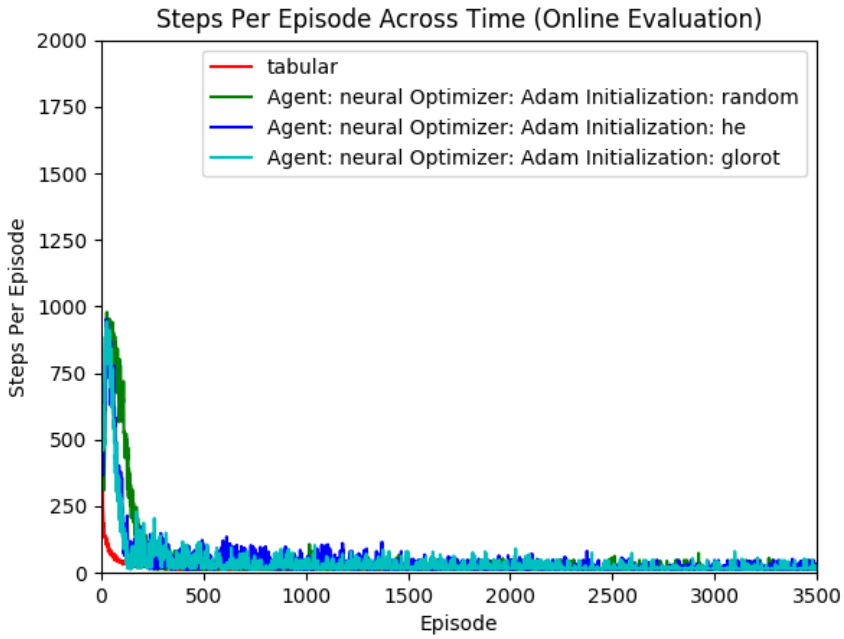

Figure 7: Online Performance 50 Neuron Architecture

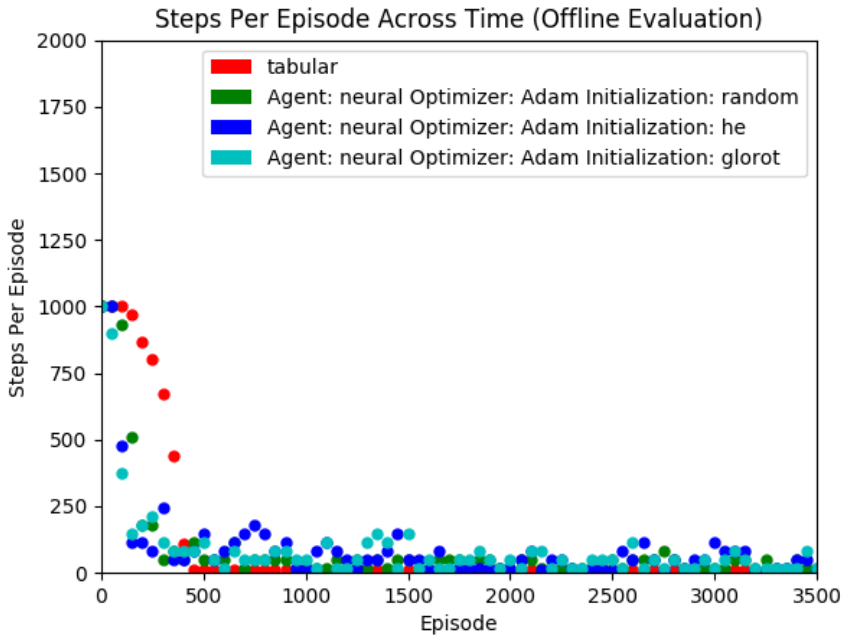

Figure 8: Offline Performance 50 Neuron Architecture

perhaps with further training the network might come to more closely resemble the tabular case, but the need to train more parameters influenced the length of time required to reach a shape similar to the tabular case.

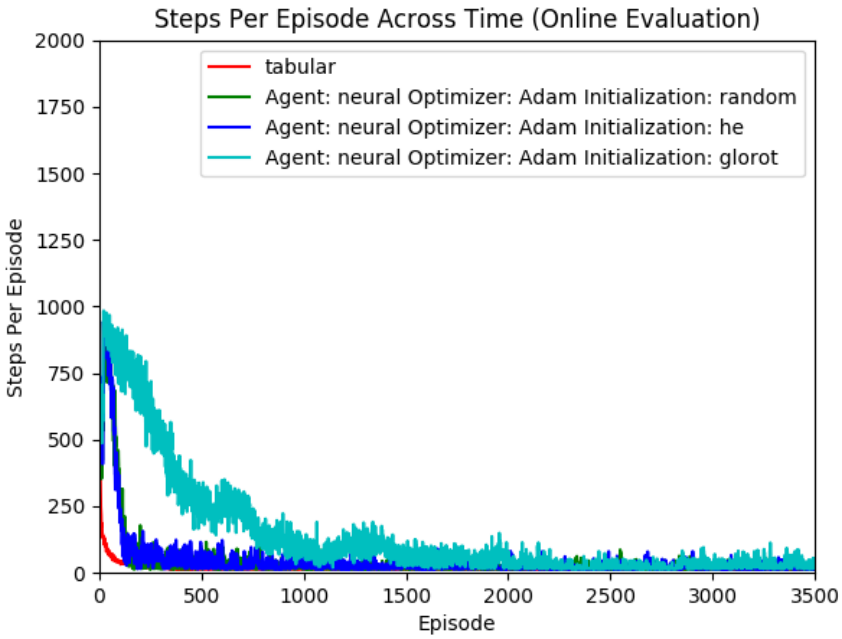

Figure 9: Online Performance 150 Neuron Architecture

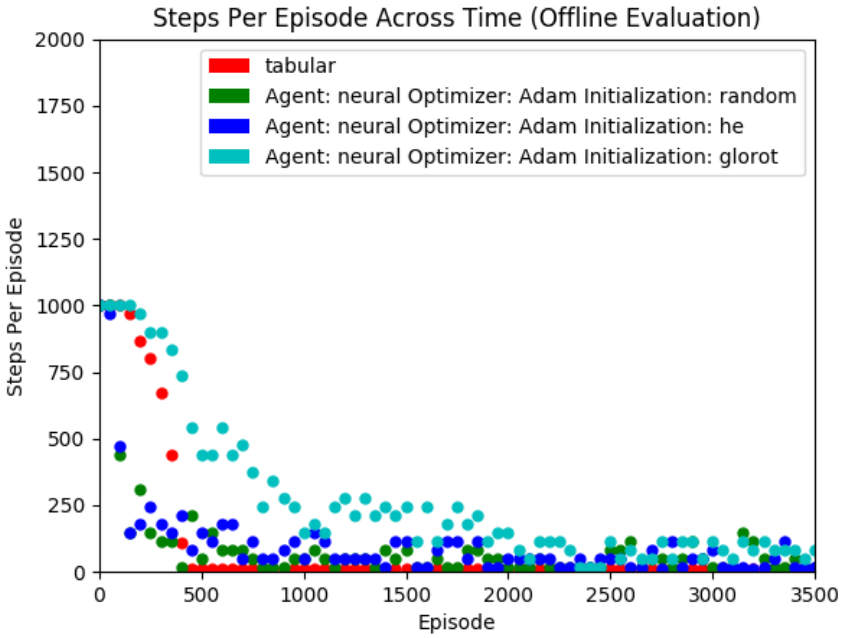

Figure 10: Offline Performance 150 Neuron Architecture

Currently, all architectures involve more parameters than there are states; as such, the network is capable of, in essence, implementing an implicit tabular representation given the 1-hot encoding for the state vector. The wider architectures necessarily have far more redundant parameters than the narrower ones, however, and it is plausible that these parameters are influencing the shape of the

value function, but in a way that does not actually add or detract from the learning of an uable value function, as evinced by the fact that the agent still peforms well enough, so long as the value of the states are correctly proportioned relative to eachother. See figures 11 to 20 for a graphical depiction of the value functions, for all network widths and initialization schemes.

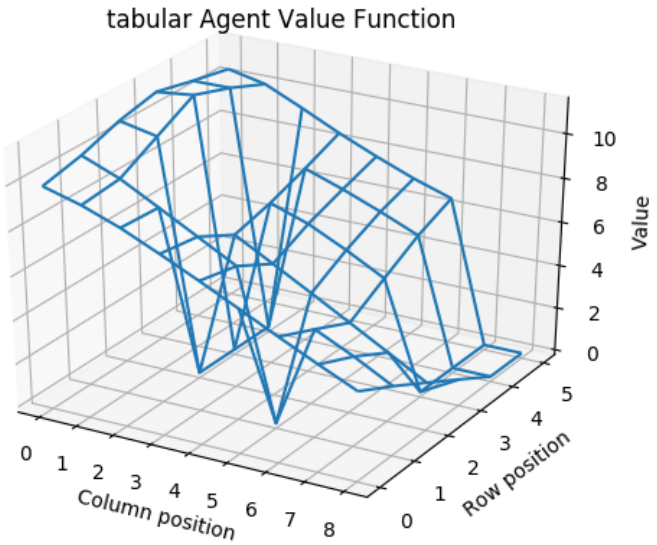

Figure 11: Tabular Value Function

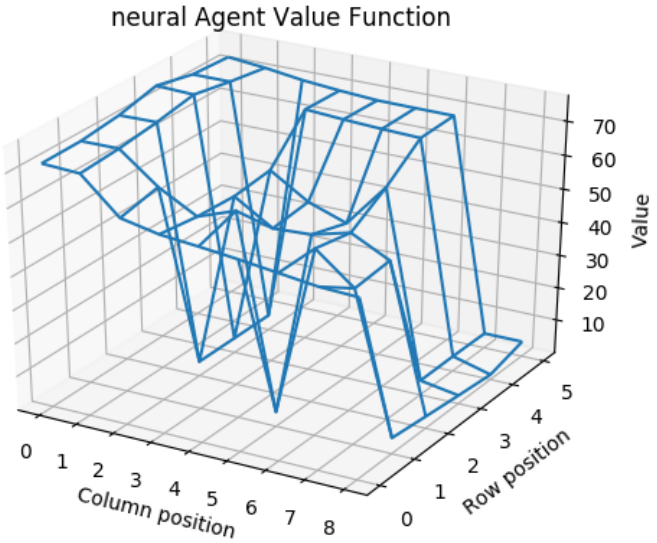

Figure 12: 5 Neuron Architecture Random Init Value Function

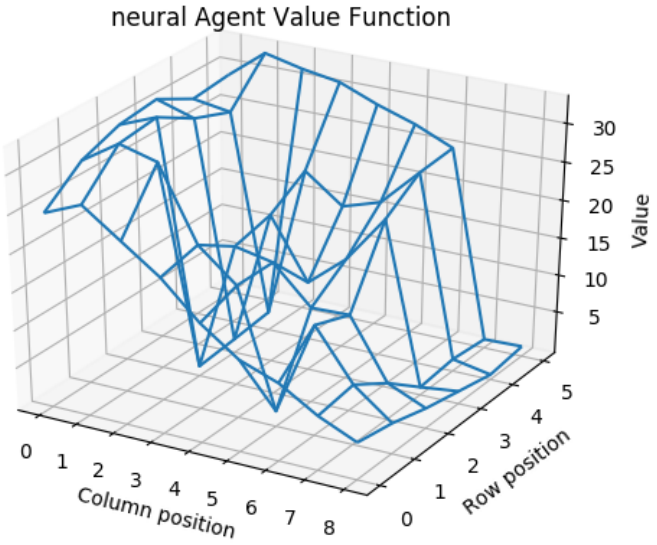

Figure 13: 5 Neuron Architecture He Init Value Function

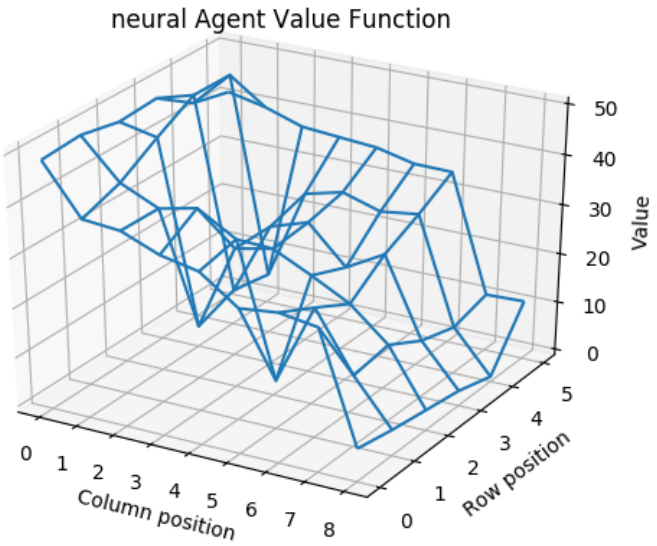

Figure 14: 5 Neuron Architecture Glorot Init Value Function

5.1.3 Visualizing The Final Layer: t-Distributed Stochastic Neighbour Embedding

t-SNE was applied to the last hidden layer of the network and, the representation reduced to 2 dimensions, for all of the states in the environment. Default parameters as provided by scikit-learn

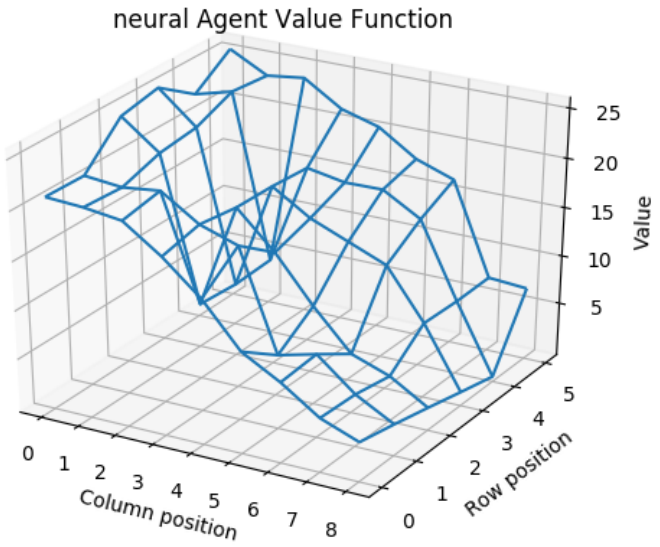

Figure 15: 5 Neuron Architecture Random Init Value Function

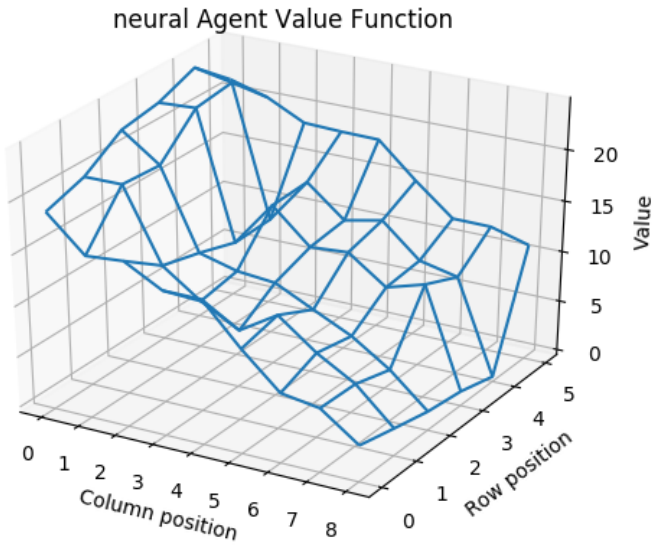

Figure 16: 50 Neuron Architecture He Init Value Function

(Pedregosa et al., 2011) were utilized for the computation of all visualizations, and the heat of each state is determined by the maximum of the state-action value for the corresponding state, as in the value function plot. The diamond represents the start state; the star; the goal state; and the squares are the states beside the walls next to the goal state in the top right corner of figure 3.

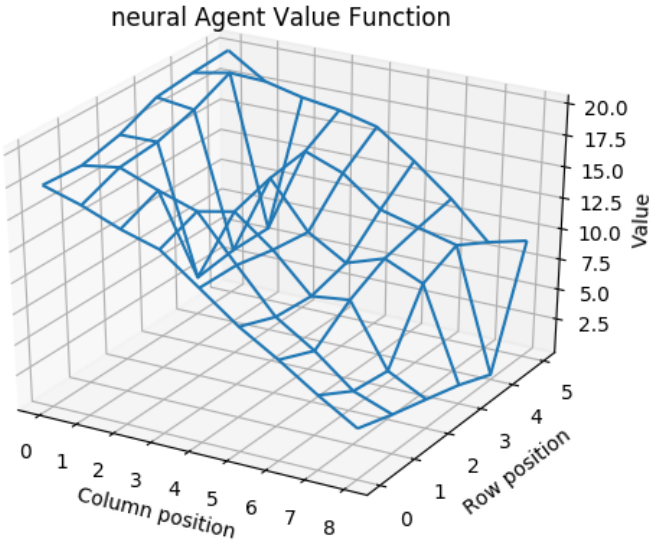

Figure 17: 50 Neuron Architecture Glorot Init Value Function

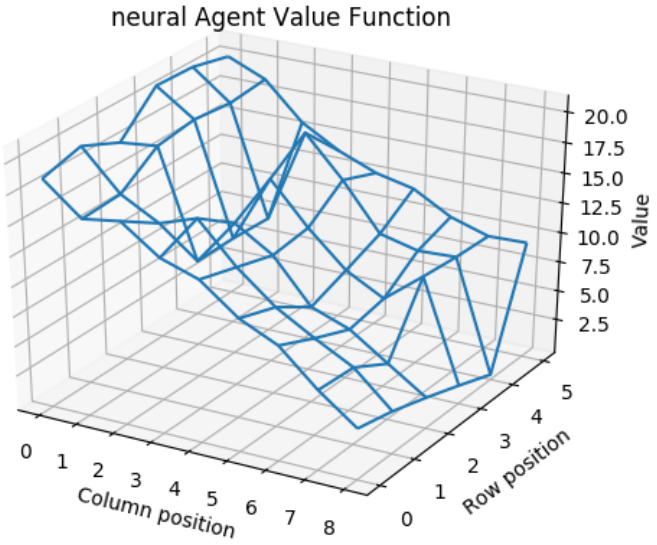

Figure 18: 150 Neuron Architecture Random Init Value Function

The reduced representations of the the hidden layer for each state vary in shape between a generic linear formation that runs diagonally across the plane, and more amorphous collections with no discernible shape. The former shape makes sense insofar as the start and goal states are in roughly diagonally opposite points in the original environment and the heatmap colouring of the t-sne gener-

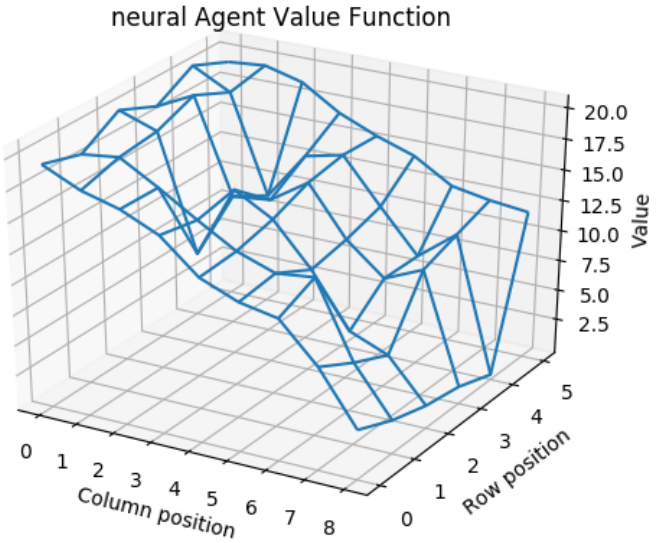

Figure 19: 150 Neuron Architecture He Init Value Function

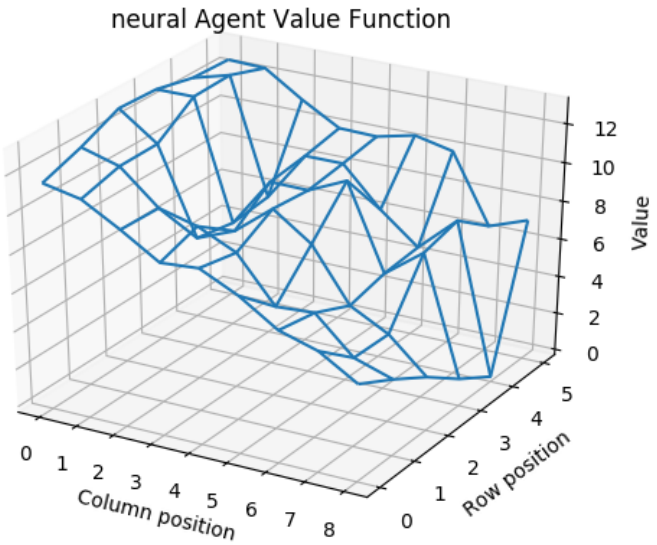

Figure 20: 150 Neuron Architecture Glorot Init Value Function

ally conforms to the value function: the points are positioned in a geomterically progressive manner that suggests the closer one gets to the state higher rewards are encountered. As well, the states next to the wall near the goal, signified by the squares, tend to cluster together in groups of 3 or more (one of them sometimes ends up position further away from the others), indicating that these states

are being recognized as similar even after being reduced back to 2 dimensions. The other pattern exhibited is less explicit in terms of shape, with the states either being spread around fairly evenly or clustering in large groups.

The 5 wide network appears to have the most consistent and roughly linear shape, while the other networks exhibit a mix of rough linear and the aformentioned cluster type patterns, depending on the initialization scheme used. The similarity of shape between the 5 wide neuron network intializations possibly reflects the fact that with 5 neurons only there is a smaller weight space, and thus less room to vary the representation in such a way so as to yield radically different two dimensional representations when run through t-sne. See figures 21 to 29 to observe the relevant visualizations for all networks and across all initialization schemes.

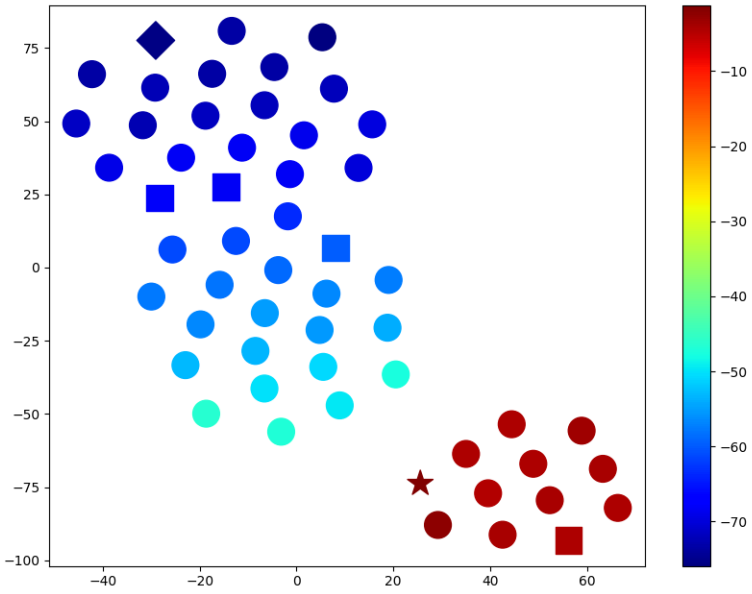

Figure 21: 5 Neuron Architecture Random Init t-SNE

5.1.4 Network Width on the Rate of Convergence of Representations

Each network had a snapshot of its architecture and weights taken every 50 episodes throughout a run, and its SVCCA similarity computed with respect to its final representation. As in t-sne, results reported are for the final hidden layer only.

The self-similarity of the network to its final representation started out quite low for the narrowest network, but increased quite sharply, for all initializations, although the he initialization appeared to converge significantly more slowly. As the network width was increased, the starting similarity of the network increased as well, overall.

This is somewhat surprising, as the larger number of parameters would seem to suggest that it would be more difficult for the networks to be similar. However, by the same token, with fewer parameters, even a small number of differences in the parameters are likely to result in a much smaller similarity score, as each neuron will in such a situation contribute more to the score; thus it is possible that significant deviations by a smaller number of neurons will yield larger differences in similarity scores.

Contrarily, with more parameters, there is more opportunity for the similarities of the distributions used in the initialization scheme to manifest themselves, in the same way that sampling more from a distibution allows for a better approximation of that distibution, setting more parameters from a particular scheme may allow for the similarities in those schemes to influence many neurons simi-

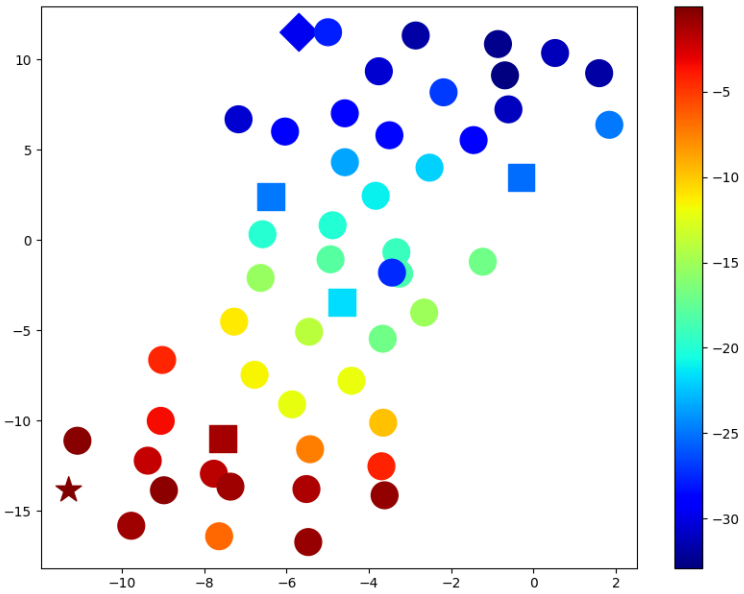

Figure 22: 5 Neuron Architecture He Init t-SNE

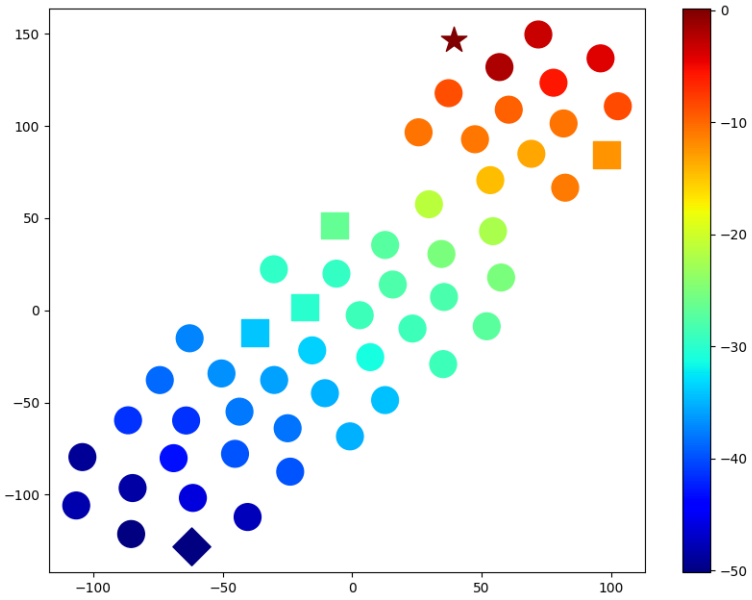

Figure 23: 5 Neuron Architecture Glorot Init t-SNE

larly, such that the differences that do occur are hidden by the similarity between larger numbers of neurons that are induced by the initialization scheme formulas.

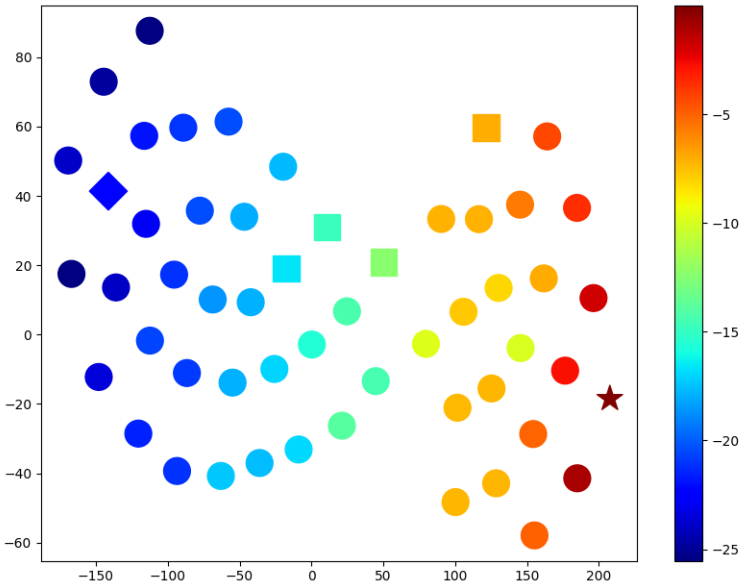

Figure 24: 50 Neuron Architecture Random Init t-SNE

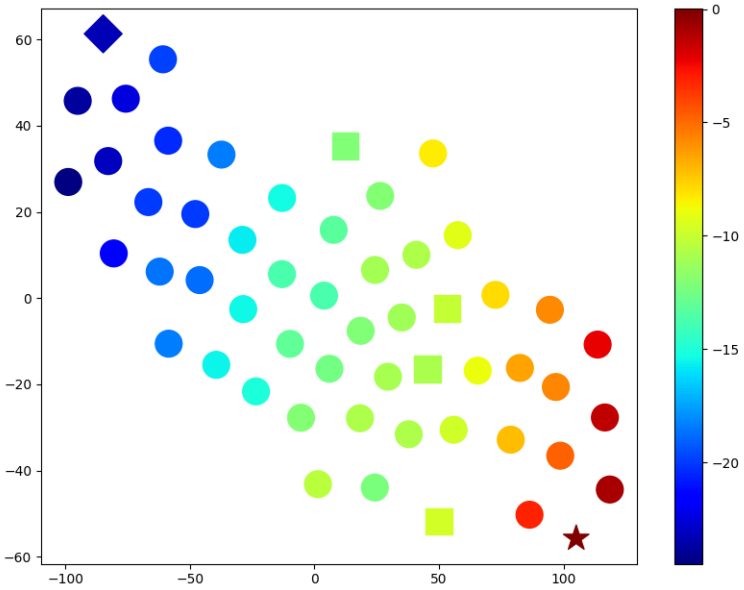

Figure 25: 50 Neuron Architecture He Init t-SNE

This increased initial similarity, however, resulted in much less change occuring over the long run for those networksn, and this appeared to have an impact on the correlation of SVCCA self-similarity with offline performance, as discuessed in the next section.

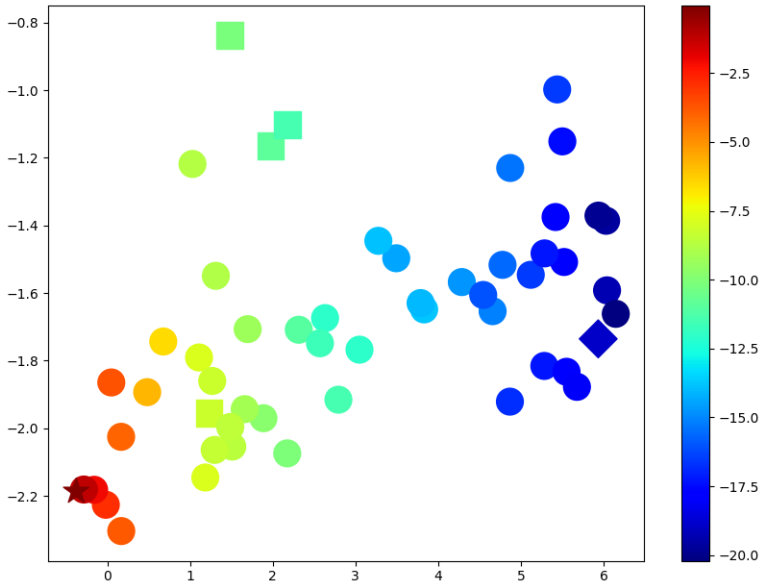

Figure 26: 50 Neuron Architecture Glorot t-SNE

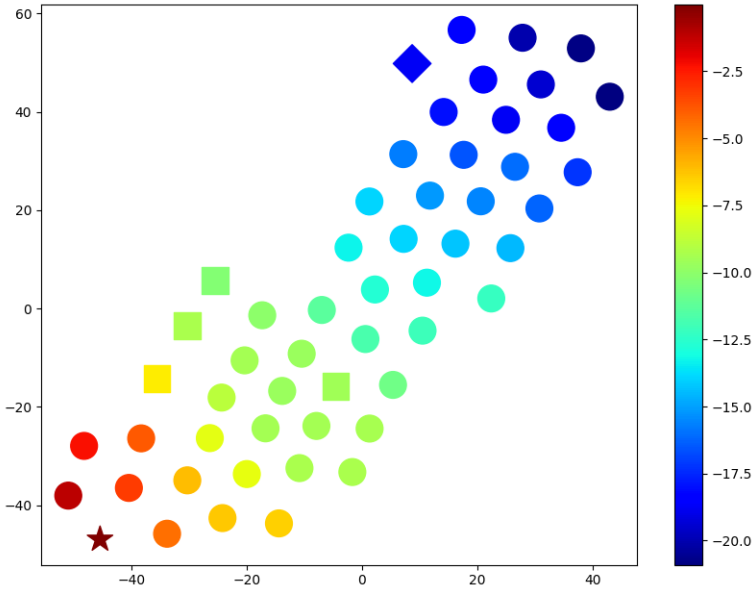

Figure 27: 150 Neuron Architecture Random Init t-SNE

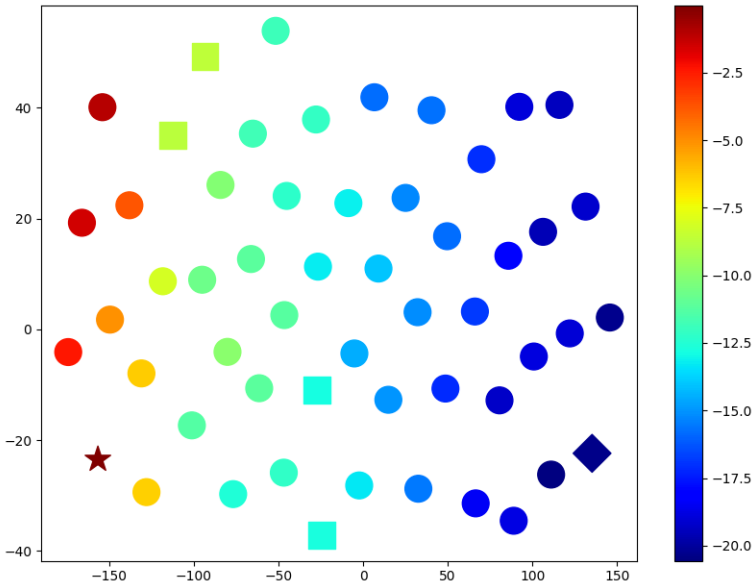

Figure 28: 150 Neuron Architecture He Init t-SNE

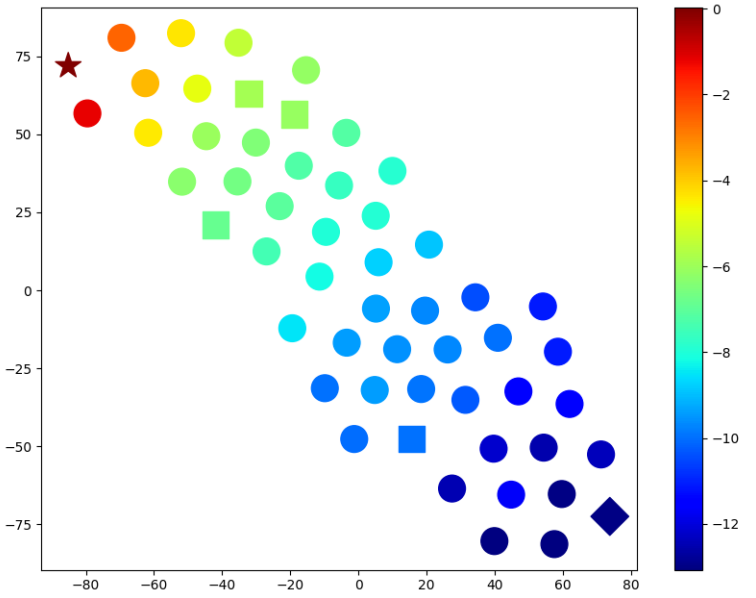

Figure 29: 150 Neuron Architecture Glorot Init t-SNE

5.1.5 Correlating Neural Representations with Performance

As mentioned, the smaller networks displayed an initial low level of similarity, but converged to their final rperesentation fairly quickly, and this is reflected in the strong negative linear and spearman cor-

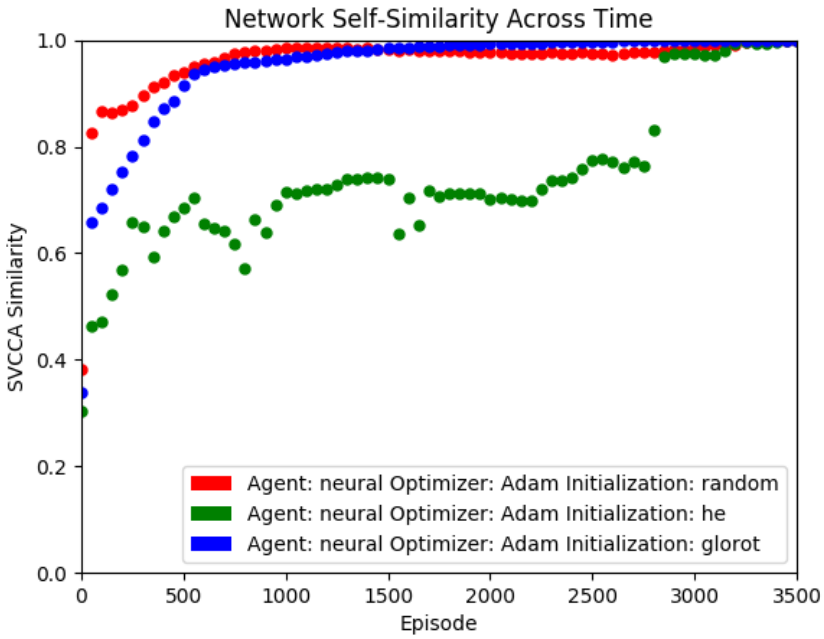

Figure 30: 5 Neurons Architecture Self-Similarity Over Time

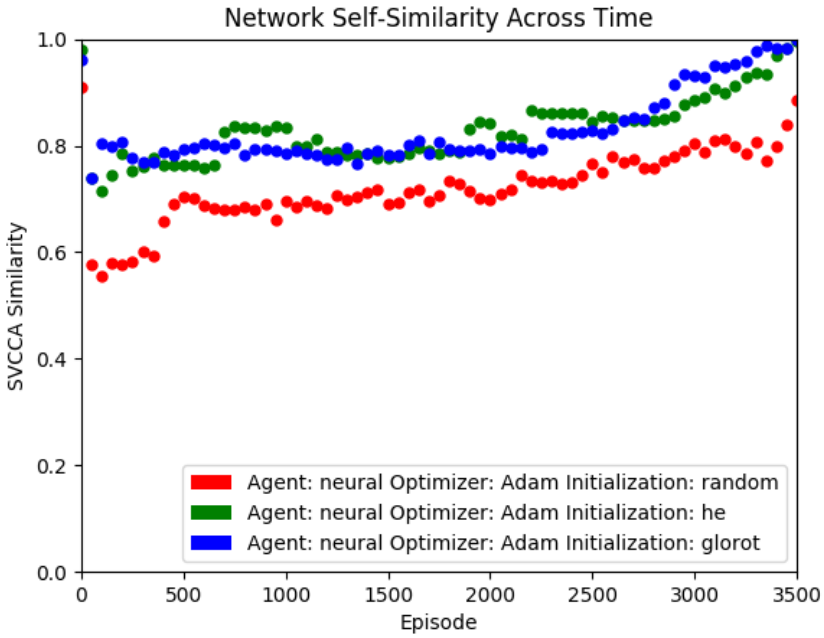

Figure 31: 50 Neurons Architecture Self-Similarity Over Time

relation coefficient between the two variables. As the network widened, however, the linear correlation weakened considerably, and eventually broke down completely, with generally non-significant p-values in the 50 and 150 wide network case. The spearman correlation remained somewhat apparent in the 50 wide network case, but it too broke down in the 150 wide network case. Insofar as the

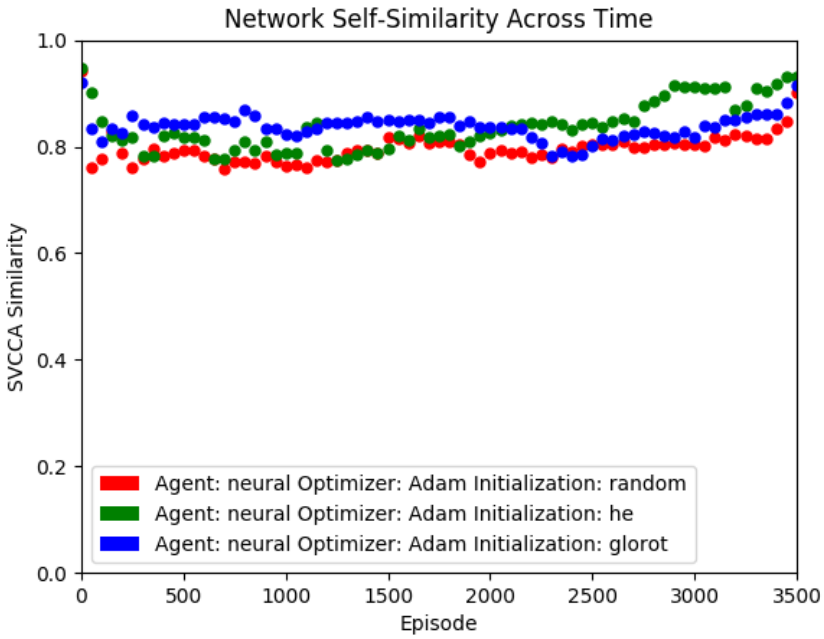

Figure 32: 150 Neurons Architecture Self-Similarity Over Time

p-values were sigificant, they tended to elucidate a negative association with offline performance, as expected, but the lack of change in the network self-similarity after initilization in the wider network cases calls into question the generality of these results. See Table 6 for a summary of the correlation results.

Nodes	Init	Pearson Coeff	Pearson P-Value	Spearman Coeff	Spearman P-Value
5	random	-0.83	3.51e-19	-0.42	3.51e-19
5	he	-0.69	1.27e-11	-0.76	1.40e-14
5	glorot	-0.86	4.83e-22	-0.90	9.55e-27
50	random	-0.24	0.042	-0.54	7.43e-07
50	he	-0.069	0.56	-0.42	0.00025
50	glorot	-0.039	0.74	-0.41	0.00027
150	random	0.10	0.37	0.024	0.83
150	he	0.17	0.15	-0.41	0.00029
150	glorot	0.22	0.062	0.28	0.015

Table 6: Pearson and Spearman Correlations Between Offline Performance and Neural Network SVCCA Self-Similarity

5.1.6 The Similarity of Convergent Representations

The similariy of the various networks last layer, at the end of the visualization run, under different initializaton schemes, are compared below.

	5 random	5 he	5 glorot
50 random	0.93	0.83	0.88
50 he	0.83	0.79	0.89
50 glorot	0.89	0.80	0.81

Table 7: SVCCA Similarity Between Network Final Layers: 50 and 5 neurons

The 50 and 5, and 50 and 150 wide networks, as expected, displayed a high to moderately high degree of similarity, as no pair of networks fell below 0.79 similarity. The 150 and 50 wide networks, however, displayed a significantly lower degree of similarity, dropping as low as 0.62 in 1 instance.

This is contrary to initial expectations, but overall the similarity measures are not so low as to call into question the validity of SVCCA, as these lower similarity appear to be something of an outlier.

Moreover, as the value function plots reveal for the current environment, only a certain degree of similarity with respect to the optimal value function is required for the agent to perform well, so it may be the case that so long as the neural network agent is performing 'good enough', there is not much pressure for the network to converge as quickly to the exact same value function, and thus corresponding representation as the best performing neural network, without running the algorithm for much longer.

	5 random	**5 he**	**5 glorot**
150 random	0.97	0.90	0.87
150 he	0.93	0.90	0.92
150 glorot	0.84	0.84	0.88

Table 8: SVCCA Similarity Between Network Final Layers: 150 and 5 neurons

	50 random	**50 he**	**50 glorot**
150 random	0.79	0.72	0.77
150 he	0.80	0.80	0.81
150 glorot	0.71	0.62	0.68

Table 9: SVCCA Similarity Between Network Final Layers: 50 and 150 neurons

5.2 Continuous Gridworld

5.2.1 Performance

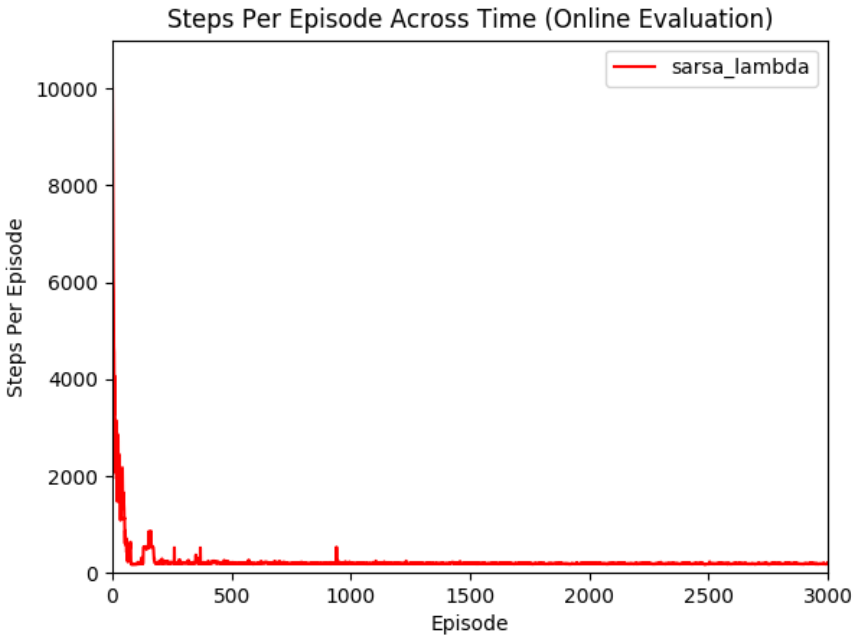

Figure 33: Continuous Sarsa Lambda Online Performance

The continuous case was quite computationally expensive to run; as such, the final results were averaged over a much smaller number of runs (3), and the results were very noisy. Moreover, the offline evaluation revealed no learning in the continuous case, even after 3000 episodes. As a result, these plots are omitted, and the sarsa lambda baseline, averaged over 30 runs, is provided alone, to provide a reference point for how well it's associated value function allows it to navigate the environment. As can be observed, sarsa lambda performs quite well, both in terms of its online and offline performance.

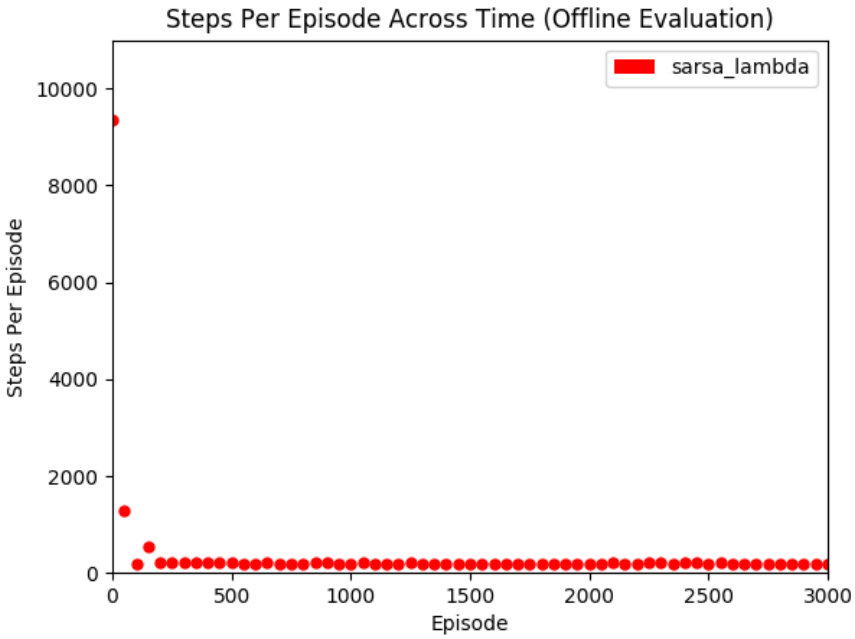

Figure 34: Continuous Sarsa Lambda Offline Performance

5.2.2 Visualizing the Learned Value Function

The learned value function for the sarsa lambda agent is quite intuitive: it has a high cost around most of the plane, but displays a tunneling effect on the cost as one heads towards the bottom right corner, where the goal state is located, reflecting that higher value states are those closer to the goal state, and lower value states are farther away. See Figure 35 for a graphical depiction of the value function as learned by sarsa lambda.

The value function for the neural network, across all initializations, vaguely approximates the same shape as the sarsa lambda value function, displaying the same tunneling effect, but much of the area is more decidedly flat than in the sarsa lambda case. These regions of the space would likely be difficult for the agent to navigate, due to the plateaux like effect, which may account for the poor performance of the agents even after 3000 episodes. However, the general shape of the value function suggests that with more training and perhaps a slower annealing rate with respect to exploration, a more similar shape to sarsa lambda would emerge.

All visualiation results are plotted with 10000 evenly spaced samples from the state space.

5.2.3 Visualizing the Final Layer: t-distributed Stochastic Neighbour Embedding

Reducing the dimensionality of the representation learned by the last layer via t-sne yields an interesting picture. None of the results display a linear clustering of shapes, although the heatmap in all cases is spread out as one would expect: many states far or moderately distanced from the goal exhibit low values, while only a smaller area, always positioned quite close to the goal state, exhibits higher values. The goal state always appears with a cluster of values close to it, while the less valuable states manifest in more varied ways, dependent upon the initialization.

The he initilization scheme (see figure 40) has the most continuity betwen states, and there is a large section that looks like what one might call a wall of unvisited states (since one cannot actually asssess the value of the positions within the wall, one cannot actually plot values for those states). The position of the goal relative to this wall shape as depicted in the t-sne graphic is on the wrong end when compared to the environment itself, however, and it is difficult to tell if this clustering

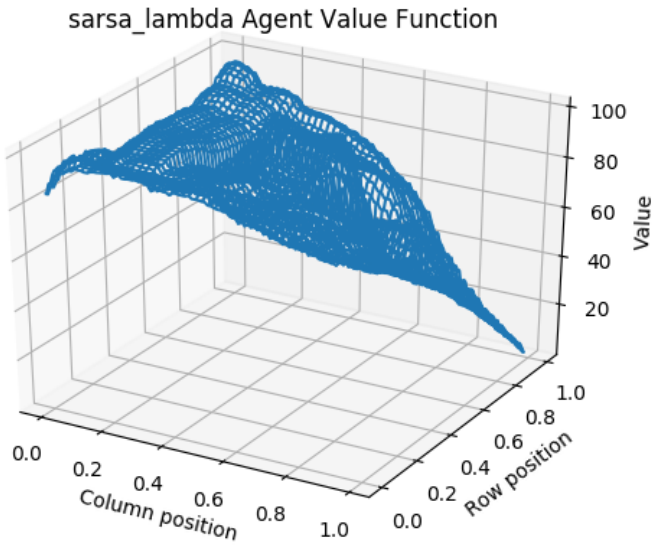

Figure 35: Continuous Sarsa Lambda Value Function

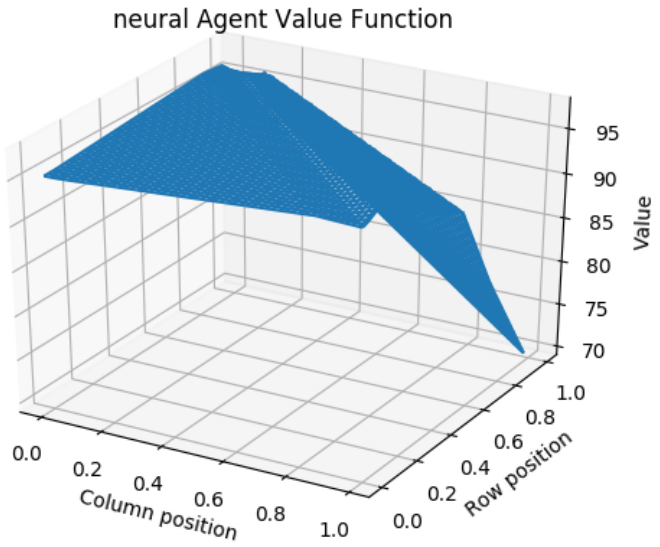

Figure 36: Continuous 150 Neuron Architecture Random Init Value Function

is simply an artifact of t-sne or whether it is a meaningful representation of the presence of the wall. The fact that the value function for the he initialization scheme (see figure 37) appears to most closely approximate the sarsa lambda value function out of all of the initialization, lends credence to

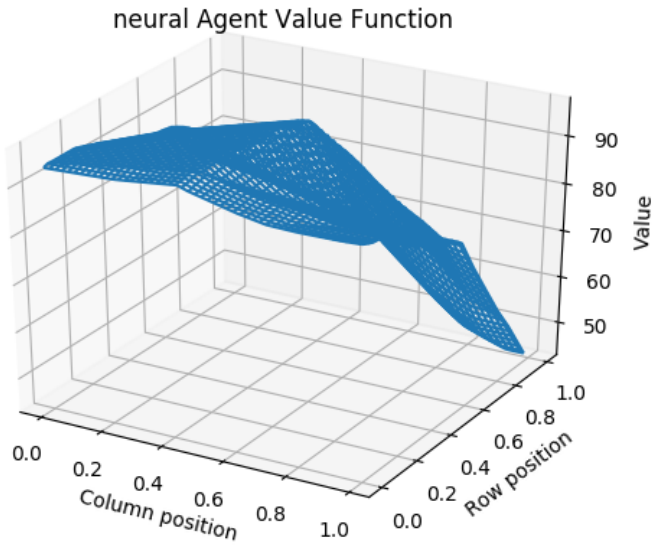

Figure 37: Continuous 150 Neuron Architecture He Init Value Function

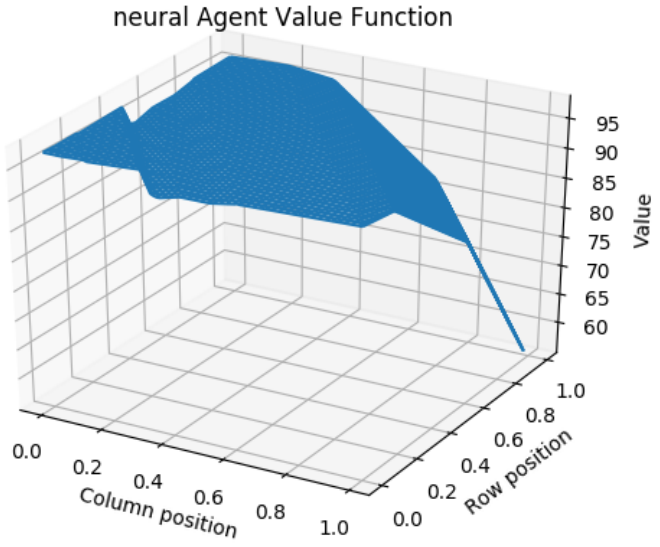

Figure 38: Continuous 150 Neuron Architecture Glorot Init Value Function

the notion that it is more accurately reflecting the value function, which is also being geometrically reflected in the corresponding t-sne plot.

Both the random and glorot initalization schemes (see figures 39 and 41, respectively) do not show any such wall pattern, and display a larger degree of clustering rather than the smooth continuity

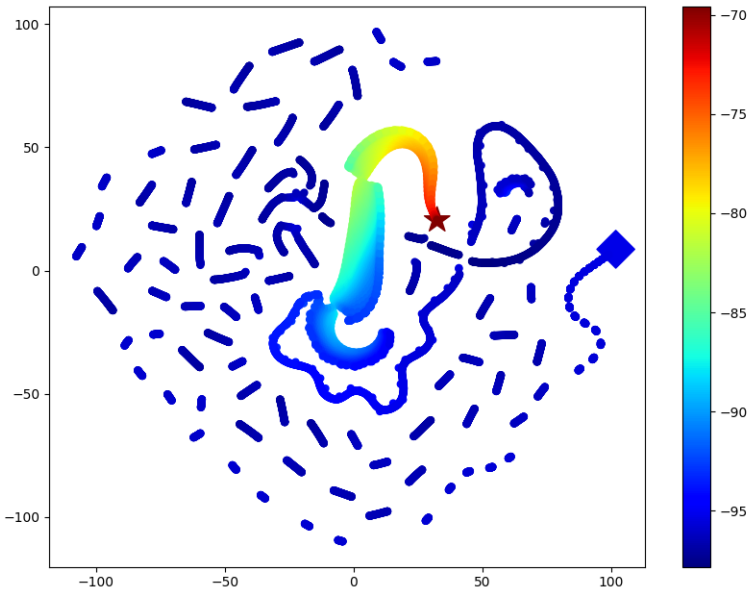

Figure 39: Random Init t-sne

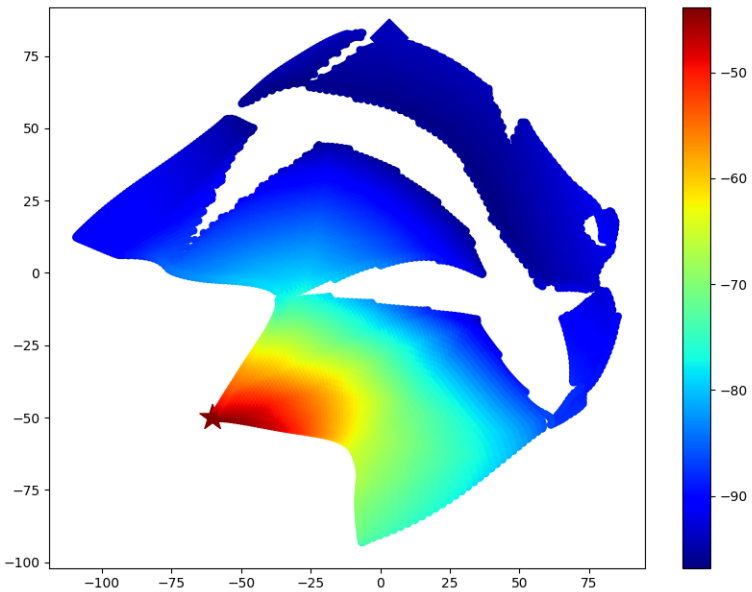

Figure 40: He Init t-sne

evinced by the he network. Each of the lines reperesents a collection of points clustered vary closely together, but these themselves are broken into spearate groups, and with different sizes, with the ran-

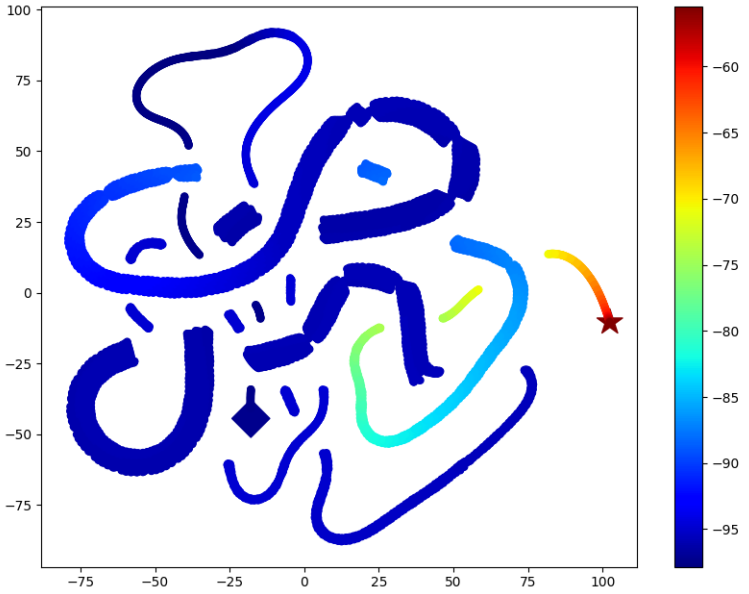

Figure 41: Glorot Init t-sne

dom initialization clustering much more frequently, but in smaller groups than in the he initialization case.

These fairly different representations are reflective of the lower SVCCA scores detailed in section 5.2.4: the glorot and he initialization are more similar to each other than either are to the random initialization, but the glorot and random initialization are still more similar than the he and random. These map onto the rough visual similarities they exhibit when reduced via t-sne, as the he initialiazation map displays the most continuity and random initializaton the least, with glorot appearing to be somewhat in between. This suggests some degree of coherency between the representation of the final neural network layer exhists between t-sne and SVCCA.

5.2.4 The Similarity of Convergent Representations via SVCCA

As the continuous case consisted of only 1 architecture, the last layer representation at the end of the run was compared for each of the three different initializations. The resultant similarity score was much lower than expected, falling well below the similarity exhibited between the different architectures used in the discrete case. This is contrary to expectations given the precedence of higher similarity in the discrete case across different architectures, and the similar value functions for the continuous case.

However, it may be that the incomplete nature of the value functions can explain some of this: given that the network appears like the value functions would converge if it were given more time to run, it may be that the same can be said of the representations as exhibited by the last layer. Indeed, it may be that given the longer time span required for convergence in the continuous case, it is possible that even after 3000 episodes the networks may be considered to still be in an comparatively 'early' phase, such that more running time would allow for convergence to more similar representations.

	he	**glorot**
random	0.29	0.37
he	X	0.64

6 CONCLUSIONS

AI has made significant progress across a number of important application domains within the last few years. Much of this can be attributed to advances in machine learning techniques and accompanying growth in the required storage and computing power over the last several decades. However, the nature of the neural network representations that underlie many of these sucessses remains theoretically inscrutable. The development of sound approaches to visualiize and interpret these representations is essential if current limitations to neural network based machine learning are to be overcome. SVCCA (Raghu et al., 2017) is a flexible formalism for comparing neural network representations that, when combined with t-sne (Maaten & Hinton, 2008), and value function visualization, can yield a considerable degree of insight into what, and how, a neural network learns within a reinforcement learning context.

Experiments in a simple discrete gridworld setting reveal that, as the network widths get wider, the value functions appear to take longer to converge to a shape similar to the optimal strategy (as represented here by the tabuar case), although this in and of itself does not necessarily reduce the performance of the agent in question, provided the value of the states relative to one another are suitably similar to the optimal case. A strong negative linear and monotonic association between offline performance and the self-simialrity of the network, at least in the case of smaller networks with fewer redundant parameters, did manifest itself. Larger networks, however, saw this correlation progressively break down as the rate of change of self-similarity with respect to their convergent representations slowed with the growth of the width of the network. Moreover, the final layer similarity for all networks ranged from high to moderately high, across all network widths and initialization schemes, which is generally in line with expectations regarding a simple environment such as gridworld, and buttresses the appropriatness of SVCCA as a good similarity measure and model of neural network layer representations.

Finally, the continuous case demonstrated what are arguably the beginnings of a similar pattern of convergence with respect to its value functions, while t-sne revealed a fair degree of coherence in terms of the visual similarity of its representation of the final layer with the similarity score as computed by SVCCA, suggesting a degree of natural complementarity between the two techinques that should be explored further.

7 LIMITATIONS & FUTURE WORK

There are a number of limitations with the current study upon which future work might improve, in terms of both rigour and scope. With respect to the former, although the discrete case is averaged over a good number of runs, the self-similarity scores used in the correlation computation are not, as they were originally intended for visualization only, which relied on a single run in order to compute those scores. Running the experiment with similar averaging for the visualization portion would ensure greater statistical validity. Relatedly, the p-values as used with the correlation coefficients are known to be somewhat unreliable for samples < 500 (Jones et al., 2001); as such, performing a correlation analysis with a greater sampling rate for both the offline performance evaluation and network self-similarity would allow for more confidence in the conclusions regarding the strength of the correlation between performance and self-similarity.

Concerning the continuous case specifically, the number of runs for the performance evaluations is quite low, at only 3 instances, due to the long training time associated with it. Performing more runs in the continuous case, for a greater number of episodes, alongside different variations in terms of both the width of the network layers and the number of layers would allow for a more rigorous and comprehensive coverage of the questions probed in the current work. Varying the annealing rate for the exploration parameter might also allow for the agent to more widely explore the state space early on in such a way so as to avoid the generation of a plateaux like value function, which could also speed up learning time and allow for faster iteration with respect to exploring the final layer representations.

Additionally, t-sne is well known to be a somewhat difficult dimensionality technique to use and interpet, due both to the (sometimes) stochastic nature of the algorithm, its sensitivity to a few hyperparameters (e.g. stopping time, perplexity, etc...), and propensity to cluster in non-meaningful ways (Wattenberg et al., 2016). The current implementation relies on sci-kit learn (Pedregosa et al.,

2011), and uses reasonable defaults for the hyperparameters, but a more thorough exploration of the hyperparameter space that integrates results from multiple plots might yield additional insights than that afforded by simply analyzing the default output of the algorithm. In addition to addressing the aforementioned issues, there is also room for expanding the scope of the current work in at least two ways. Firstly, Morcos et al. (2018) have since recently modified SVCCA to address some of its shortcomings as a measure of representational similarity. Taking this new technique and applying it within the context of a similar study would be useful insofar as it would allow for an assesment of the benefits of the alleged improvements to the method, in a simple environmental context that would allow for the easy progressive modification of the environment to test at what point the touted benefits of PWCCA begin to accure, if they do not already show a significant difference in the current environment under consideration.

Lastly, taking one or both of these techniques (t-sne and SVCCA) and applying them within a transfer learning or auxiliary task learning context could prove very insightful. Both of these scenarios are ones in which there are multiple factors beyond the main task influencing the representation process, either simultaneously, as in the case of auxiliary task learning, or sequentially, as in transfer learing; and it is likely that there are a number of fruitful questions regarding the dynamics of learning that could be posed and answered in these particular cases with the help of t-sne, SVCCA, and the relevant value functon plots.

8 Acknowledgments

The author would like to thank Adam White, Cam Linke, and Taher Jafferjee for their commentary and constructive conversation concerning the present work, and both the Department of Computing Science at the University of Alberta, and Cybera Incorporated, for providing access to computational resources.

References

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pp. 1–8, 2007.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 262–275. Springer, 2017.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Institute of Electrical and Electronics Engineers (IEEE) transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Serkan Cabi, Sergio Gomez Colmenarejo, Matthew W. Hoffman, Misha Denil, Ziyu Wang, and Nando de Freitas. The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. In *1st Conference on Robot Learning (CoRL)*, 2017.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pp. 216–217, 2008.

François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

George Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192, 1989.

Francis Galton. Co-relations and their measurement, chiefly from anthropometric data. *Proceedings of the Royal Society of London*, 45(273-279):135–145, 1889.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*, 2015.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649. Institute for Electrical and Electronics Engineers, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the Institute for Electrical and Electronics Engineers (IEEE) international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pp. 857–864, 2003.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3): 90–95, 2007.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, April 2017.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`.

Diederik Kinga and Jimmy Ba. A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, volume 5, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? In Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar (eds.), *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pp. 196–212, Montreal, Canada, 11 Dec 2015. PMLR. URL http://proceedings.mlr.press/v44/li15convergent.html.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Jose F Martinez and Engin Ipek. Dynamic multicore resource management: A machine learning approach. *Institute of Electrical and Electronics Engineers (IEEE) Micro*, 29(5), 2009.

Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pp. 51–56. Austin, TX, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, 86(1):82–85, 1998.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 5732–5741. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7815-insights-on-representational-similarity-in-neural-networks-with-canonical-cor
pdf.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pp. 6076–6085, 2017.

Herbert Robbins and Sutton Munro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–354, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404. URL http://science.sciencemag.org/content/362/6419/1140.

Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

Richard S. Sutton. tiles3. http://incompleteideas.net/tiles/tiles3.html, 2018.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 2018.

Brian Tanner and Adam White. Rl-glue: Language-independent software for reinforcement-learning experiments. In *Journal Of Machine Learning Research*, pp. 2133–2136, 2009.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 9607–9616. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8167-towards-understanding-learning-representations-to-what-extent-do-different-ne
pdf.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Martin Wattenberg, Fernanda Vigas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016. doi: 10.23915/distill.00002. URL http://distill.pub/2016/misread-tsne.

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Shangtong Zhang and Richard S. Sutton. A deeper look at experience replay. *Computing Research Repository (CoRR)*, abs/1712.01275, 2017. URL http://arxiv.org/abs/1712.01275.