M2VEC: MDP EMBEDDINGS THROUGH NLP

Ryan Tolsma, Drew Gregory, Daniel Tan, Kendrick Shen

{rtolsma, drewgreg, dtch1997, kshen6}@stanford.edu

Introduction

Markov Decision Processes (MDPs) have become a common tool for modeling stochastic decision tasks, and as such it is of great interest to develop an optimal policy-finder via reinforcement learning (RL). However, traditional tabular methods for finding such a policy of an MDP do not scale well with MDP size, and more innovative approaches to combat this limitation, such as deep Q-learning, have been explored. As with any function approximator, feature engineering for deep Q-learning is generally difficult, and in this project we explore automated feature extraction and subsequent optimal policy-finding on MDPs via Word2Vec, a natural language embedding method.

MDP Embeddings

We use a random uniform policy (at each state, randomly choose an action) to simulate walks on a synthetically-generated MDP. Each walk is a “sentence”, consisting of state-action pair “words.” These “sentences” are fed into Word2Vec to obtain embeddings of each state-action pair in $\mathbb{R}^m$. For our purposes, $m = 10$. Internally, Word2Vec trains a neural network on a skip-gram model to learn embeddings that maximize the ability for embedding inner products to represent joint co-occurence and continuous-bag-of-word probabilities.

Dataset

We generated synthetic MDPs to use as our dataset. For each state-action pair, transition probabilities to the other states were drawn independently and uniformly from $[0, 1]$ then normalized to sum to 1. Rewards were generated as independent and identically-distributed standard Gaussian variables.

Clustering of Embedded Vectors

Various MDP types were explored and embeddings compared to measure the ability of Word2Vec to recover latent structures. In all cases, embeddings of state-action pairs generally clustered by state (using a Euclidean distance metric), suggesting that they do capture real information about each given MDP. Sample results are shown below.

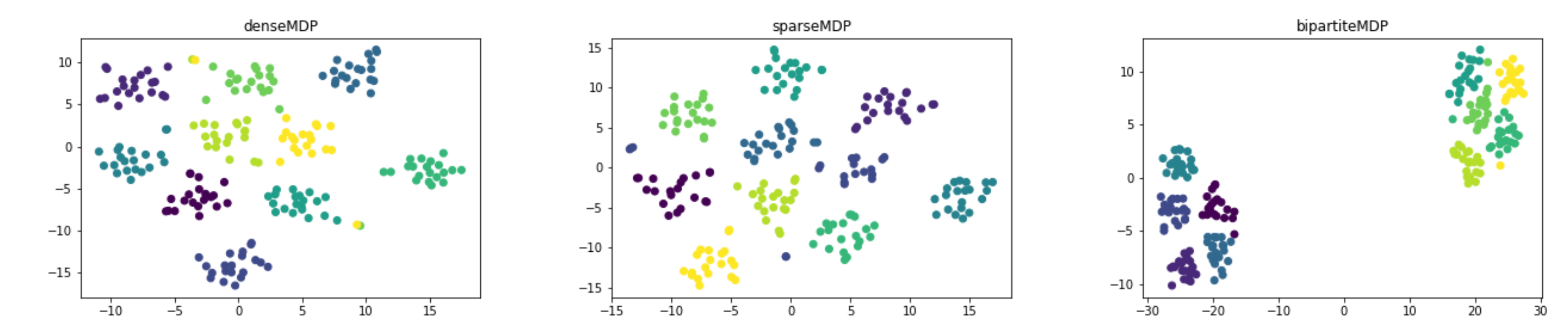

Fig. 1: Plot of state-action pairs for each MDP, labelled by state. For this experiment, MDPs had 10 states and 20 actions. t-Distributed Stochastic Neighbor Embedding (t-SNE) was used to map embedded vectors into $\mathbb{R}^2$

Embedding Dimension Validation

In order to validate the choice of embedding dimension, we ran k-means on the embedded vectors and plotted graphs of the loss (average squared distance of a point to the center of its cluster) for various values of k. Since the state-action pairs cluster by state, the ”best” embedding dimension m should be the one where the elbow point of the graph occurs close to the number of states. For MDPs of size 100, we found that this occurs at around $m = 15$, and so we used this value for all other experiments.

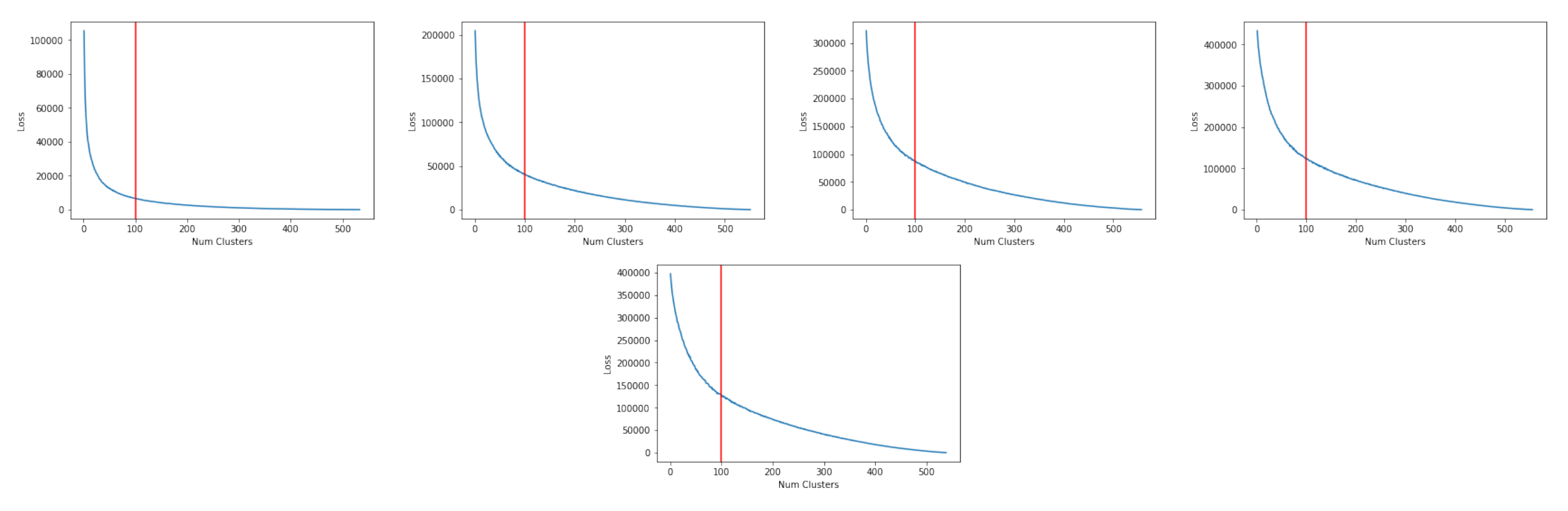

Fig. 2: Plot of average squared distance to center against value of k used in k-means. From left to right, $k = 5, 10, 15, 20, 25$. For this experiment, the MDPs had 100 states.

Deep Q-learning Network

For each MDP, a deep Q-learning network was trained using the embeddings as features. The aim of this DQN is to predict the expected value of each state-action pair under the optimal policy. To measure the success of this approach, we compared these values to the values learned through value iteration. The values correlated positively with Pearson’s correlation coefficient $\rho = 0.05766$.

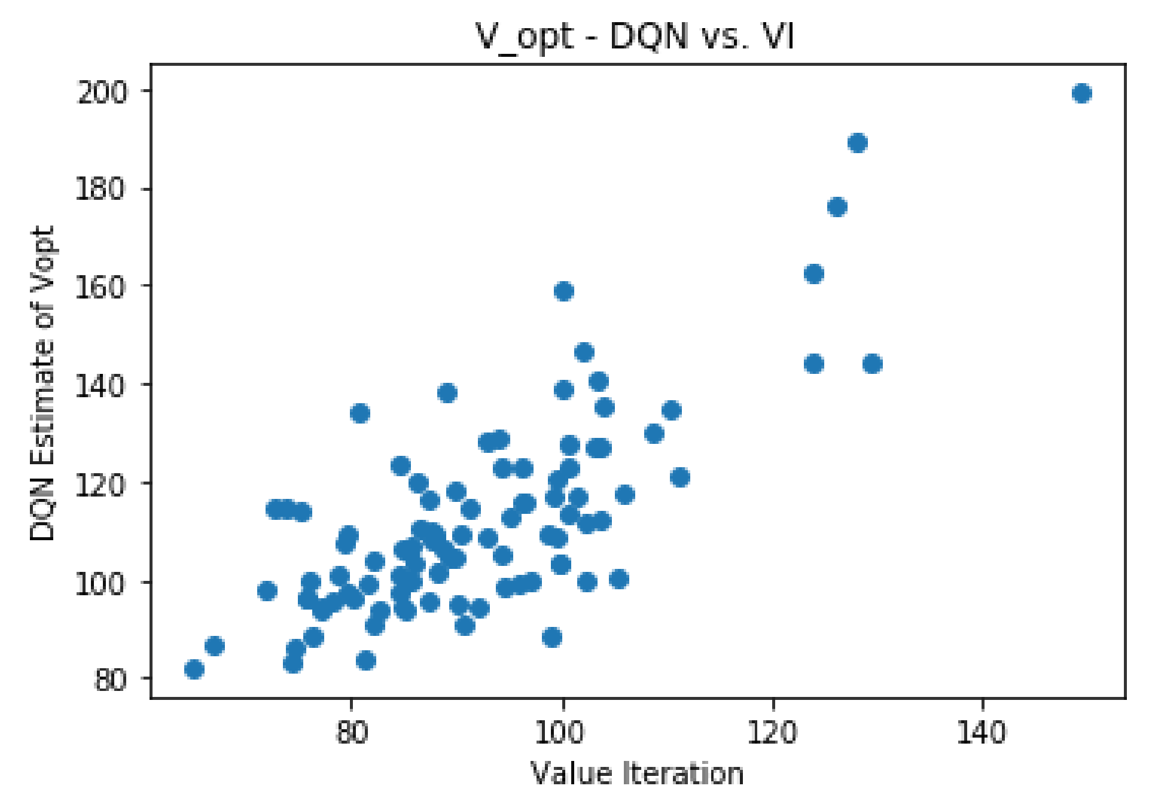

Fig. 3: Scatter plot of DQN-learned values versus true values

We evaluated our embeddings performance with a DQN against value iteration that had knowledge of the transition probabilities. We realized that there is a time/accuracy tradeoff that can be used to compare this new model against more naive algorithms like Model-Based Monte Carlo, which would require estimating all the transition probabilities. We also noticed that our random walks required to generate these embeddings could also be leveraged to estimate the MDP and give the evaluation function an improved starting target.

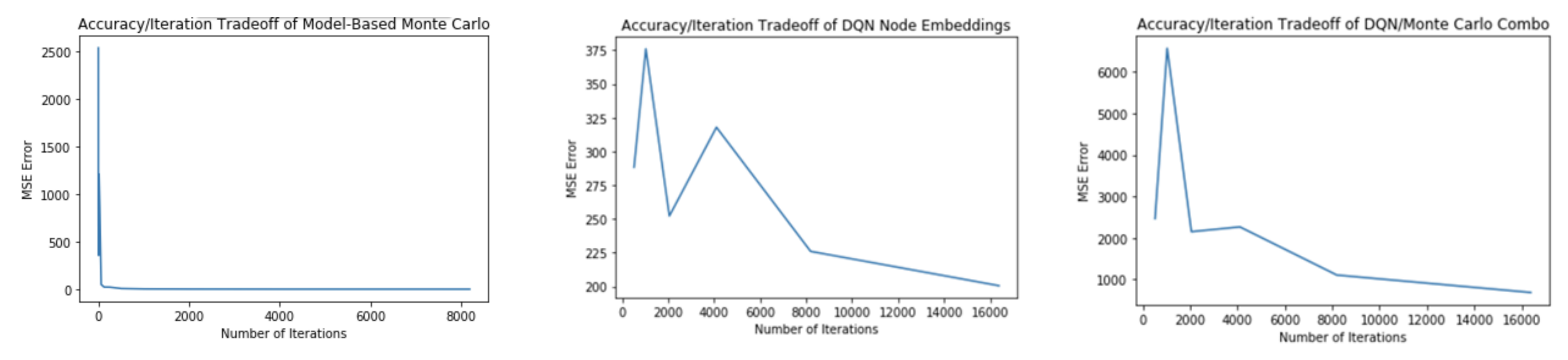

Fig. 4: Mean squared error of state values against number of iterations for various algorithms. Here, the Y-axis is the mean squared error with respect to the ground truth state values learned by value iteration