BRAINIARB: BEHAVIORAL RECONSTRUCTION AND INSIGHT VIA NEUROETHOLOGICAL IMITATION AND REALISTIC BODIES

A08 Group
Halıcıoğlu Data Science Institute
University of California San Diego
La Jolla, CA, USA

Eric Leonardis & Talmo Pereira
Salk Institute for Biological Studies
La Jolla, CA, USA
talmo@salk.edu

ABSTRACT

Simulations of animal behavior have the potential to reveal new insights into the mappings between the brain and natural behaviors that animals evolved to produce. Artificial neural networks applied in the context of task learning have recently been used as a common model of animal behavior. We reproduce various visualization and network comparison techniques traditionally applied to neuroscience to understand artificial neural networks, trained via offline and online reinforcement algorithms, in the context of biologically realistic behavior.

1 INTRODUCTION

Living animals are one of the primary sources of data for the development of artificial neural systems. In general, animals are able to perform all of their behaviors and actions under the guidance of a single neural network and its underlying algorithms. One of the end goals for artificial intelligence systems is to emulate these complex neural networks in a virtual manner, to allow our creations to act in a way that replicates how a real animal would behave and learn. Embodied control in neuroscience holds the potential to deepen our understanding of how the real brain works. If a sufficient model of the brain can be created in conjunction with the behaviors that a biorealistically simulated agent generates, then we can quickly generate new hypotheses about real world mappings between the brain and behavior. To achieve this researchers within this field are working to decode all the parts of animals' neural networks through inferential and replicatory techniques. One of the leading method to do so is to build models to learn how to behave like a real animal, mainly through deep learning models that emulate imitation learning or reinforcement learning. Prior research within this intersection of neuroscience and machine learning have explored methods to solve tasks given input data from a sensory channel, such as vision or sound. The particular approach taken in "Deep Neuroethology of a Virtual Rodent", along with this replicatory work, focuses on understanding the underlying mechanisms of "embodied control" Merel et al. (2019), which is how animals utilize a "combination of their body, sensory input, and behavior" to accomplish tasks, with a particular focus on motor control. The implications of this study include furthering developments in understanding the underlying mechanisms of animal behavior within the fields of motor neuroscience and artificial intelligence.

Since all the underlying mechanisms and algorithms for a live animal's neural network cannot be easily observed, it is difficult for observers to understand exactly what is happening. Replication with deep learning models, however, gives us a chance to pull out all the data from the agent and see what exactly is happening within the simulated neural network. In addition to this, we are also able to observe and control nearly every feature of both the agent and environment with the use of three-dimensional physics simulators. The virtual nature of such models give us significantly more flexibility compared to experiments with live animals. From this data, inferences can be made to give us an understanding of what policies go into play during certain actions or behaviors.

Our work builds off the technical framework in Merel et al. (2019), where they use the MuJoCo physics simulation introduced by Todorov et al. (2012) to simulate a biorealistic rodent model, created with a skeleton and muscle groups from dissections of a rat. Using an artificial neural network as a model for the brain, this rat is trained to accomplish various tasks such as foraging for food, navigating a maze, and avoiding obstacles - taking sensory information like a real rat would and using it to actuate muscles in its body in order to accomplish the tasks at hand. In an extension of Merel's groundwork we utilized a simplified humanoid model that can perform a walking action as a human would. In addition to the online reinforcement learning methods utilized in Merel's work, we expanded on the algorithms used to train a second simulation based on offline reinforcement methods. With these trained models in hand, the neural networks can be extracted and explored for the difference and similarities between the different reinforcement learning frameworks.

The authors apply various visualizations over timescale, kinematics, and neuronal population dynamics in order to understand the behavior and neural dynamics of a rodent in the same manner as a neuroscientist would. Classical methods such as tSNE and PCA have been used van der Maaten & Hinton (2008), and methods to visualize rotational and oscillatory dynamics of complex models have been utilized, such as jPCA (Churchland et al. (2012)) and Centered Kernel Alignment Nguyen et al. (2020). In our work, we seek to reproduce visualization techniques drawn from neuroscience. We plan to borrow techniques focused on visualizing neural populations to analyze the populations of neurons within artificial neural networks akin to real brains. Overall, the goal in this work is to evaluate the differences and similarities between models trained via online reinforcement learning and offline reinforcement learning. This replication, it will give us further insight into the nuances between the two.

2 METHODOLOGY

The models used for this evaluation are based on the framework for the Walker2D agent from the Mujoco Gym environment. As per the Gym documentation, the walker builds upon the prior hopper environment, whose task is to jump on one leg, by adding an additional leg to allow for walking movements. The model is composed of a torso, two upper leg segments, two lower leg segments, and two feet, as shown in Figure 1.

The task used for evaluation was the walking task, which involves the agent learning how to apply torque on the joints to make itself walk forward without its position moving out of the specified bounds of the environment or its joint angles being outside a specified range. The training process involves teaching the agent how to act through both online and offline reinforcement learning where the walker is rewarded and penalized according to its reward function. The reward is calculated through three components: the healthy reward, the forward reward, and the control cost. The healthy reward is a fixed value awarded to the agent every timestep if it is upright and moving faster than a specified speed. The forward reward is measured by the forward reward weight times the difference between the x coordinate before the action and the x coordinate after the action, divided by the time between actions. The control cost is a penalty if the agent makes an action that deviates too much from the previous one. The total reward is calculated by adding the healthy reward bonus and forward reward then penalizing the agent for the control cost by subtracting it.

Torque can be applied to any combination of the upper leg, lower leg, or foot for the walker's legs. From these models, we can extract data–such as joint angles, joint velocities, and positional data–from the available observation space for later comparison between the model itself or the two differently trained models.

The two algorithms used to train an individual agent are online reinforcement learning and offline reinforcement learning for the purpose of exploring the representational differences within the respective neural networks. The main difference between online and offline methods involves what data is made available to the agent at any given time step and how it learns from it through the training.

The first walker agent will be trained through online reinforcement learning, where the agent will be learning how to perform its task, walking, through interactions with its environment. Specifically, the Soft-Actor Critic reinforcement learning algorithm guides the agent's decisions and interactions with the environment. This is an off-policy algorithm that optimizes both a Q-function and stochastic policy function at each timestep by using the policy gradient algorithm, along with utilizing entropy regularization to promote the exploration of a broader range of actions. The "critic" network of the Soft-Actor Critic (SAC) algorithm takes in the state and action and outputs the Q-value from the Q-

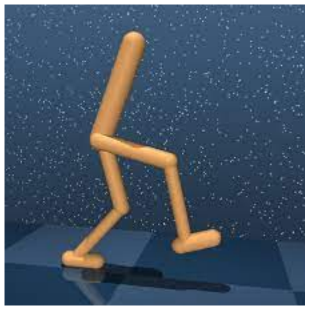


Figure 1: Walker2D Agent

Observation	Measurement (Units)
z-coordinate of the top (height of hopper)	position (m)
Angle of the top	angle (rad)
Angle of the thigh joint	angle (rad)
Angle of the leg joint	angle (rad)
Angle of the foot joint	angle (rad)
Angle of the left thigh joint	angle (rad)
Angle of the left leg joint	angle (rad)
Angle of the left foot joint	angle (rad)
Velocity of the x-coordinate of the top	velocity (m/s)
Velocity of the z-coordinate (height) of the top	velocity (m/s)
Angular velocity of the angle of the top	angular velocity (rad/s)
Angular velocity of the right thigh hinge	angular velocity (rad/s)
Angular velocity of the right leg hinge	angular velocity (rad/s)
Angular velocity of the right foot hinge	angular velocity (rad/s)
Angular velocity of the left thigh hinge	angular velocity (rad/s)
Angular velocity of the left leg hinge	angular velocity (rad/s)
Angular velocity of the left foot hinge	angular velocity (rad/s)

Table 1: Summary of Walker2D Recorded Kinematic and Geometric Measurements

function. The "actor" network learns to optimize the policy function by taking the state as the input and outputting the probability distribution over the action space. This actor-network is supervised by the critic network to minimize the mean squared error between the predicted and actual Q-values. Online reinforcement methods allow for adaptability in the agent's actions due to it adjusting its policies accordingly given the environment state in real-time. However, this means that the agent will essentially be learning through trial and error, so it will be making suboptimal actions in the training process.

The second walker model will be trained through offline reinforcement learning methods, meaning that the entire dataset of the collected observations in Table 1, along with the action and reward states, will be fed to the agent to train upon. This reinforcement learning method involves training the agent on pre-collected data, in this case, data from the agent trained via online reinforcement learning. Since all the data is available to the agent upfront, no interactions with the environment are needed. Once the "expert" demonstration is trained, a second agent can be trained via behavior cloning by learning how to imitate the actions of the online reinforcement learning Walker given the expert data. Behavior cloning is a supervised learning method where the predicted actions are compared against the expert demonstration and, eventually, learns a policy through maximum likelihood estimation.

One limitation of offline reinforcement learning is that it does not generalize well to situations not found within the training dataset. Unlike online reinforcement learning, agents trained via offline methods do not interact with the environment in the training process, so it will only be trained on situations within the expert demonstration's training dataset. This can result in poor performance in scenarios not present in the training set and non-robust policies. To address this challenge, the noise was added to the training dataset for the behavioral cloning model. From this, eight variations of the same model were generated to explore the effect noise had on how well the offline model could learn and generalize. Models for the following amount of noise were created and used for comparison: 0%, 5%, 10%, 20%, 40%, 80%, and 160%. The addition of noise is especially beneficial for offline models since it makes the data more diverse and generates additional situations, which in turn helps the trained policy generalize to new data and unexpected situations.

On paper, online reinforcement learning boasts adaptability to environmental conditions. In contrast, offline reinforcement is more efficient in learning tasks since it does not require interactions with the environment and does not face the exploration issues that online reinforcement learning methods typically do. The capabilities of the two algorithms will be tested and analyzed for differences in performance and representational similarity.

After training the model, we can now compare the similarity between the neural activities of the two differently trained agents. Central Kernel Alignment is the primary method used to compare the representational similarity of the neural networks of the offline and online agents. The CKA similarity metric was calculated using the methodology described in by Simon Kornblith (2019).

$$\mathrm{CKA}(\mathrm{XX}^{\mathrm{T}}, \mathrm{YY}^{\mathrm{T}}) = \frac{\|XY^{\mathrm{T}}\|_{\mathrm{F}}}{\|XX^{\mathrm{T}}\|_{\mathrm{F}}\|YY^{\mathrm{T}}\|_{\mathrm{F}}}$$

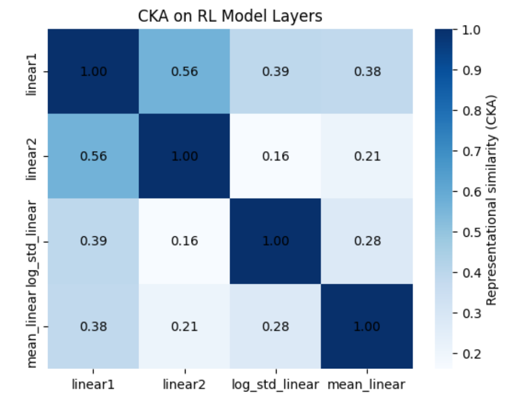
In the CKA equation formulated by Simon Kornblith (2019), $\|*\|_{\mathrm{F}}$ is the Frobenius norm. CKA compares the representational similarity of features between neural networks to compute a similarity metric. It does so by calculating the similarity between the matrix of pairwise similarities for each set of features and then finding the normalized Frobenius inner product between the pairwise similarity of the central subspaces of the feature sets after centering them. The result is a value between 0 and 1, with 0 signifying no similarity and 1 meaning the features are exactly the same. The representational similarity of each pairwise combination of activation layers within the SAC and behavioral cloning model can be compared between the two networks and within each individual network.

In order to compare the walking behavior of the two agents, we use t-distributed Stochastic Neighbor Embedding (t-SNE) as a dimensionality reduction method for mapping the agent's 3D movements through time (consisting of height positions of every body part for each timestep) into 2D space. The mapping produces distinct clusters/groupings, which shows how the agent's gait changes as it moves through time.

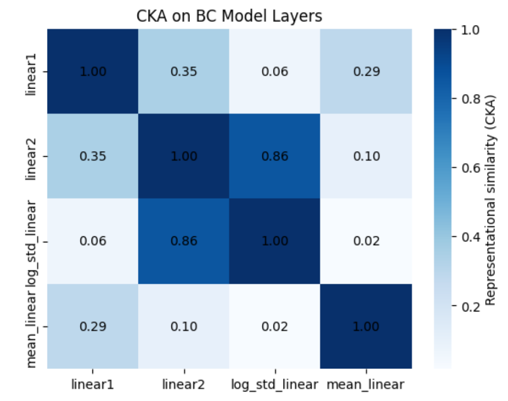
The process of mapping the height positions of various body parts of the agent, with respect to the ground, into 2D space involves the calculation of similarities of each input data point (a series of height positions at each timestep) to every other input point in the collected agent kinematics. t-SNE then attempts to learn a 2-dimensional map that reflects these similarities as well as possible. This mapping is determined by minimizing the KL divergence of the distribution of similarities in the original input space with the distribution of similarities between mapped points in the output 2D space (Maaten Hintonet al.).

The clusters inherent to a t-SNE plot are better pronounced by applying a watershed to the plot. A watershed clustering algorithm applied to this mapping, which uses KDE to highlight central ”hotspot” regions within the t-SNE, is expected to identify key groupings between different gaits adopted in the agent's movement.

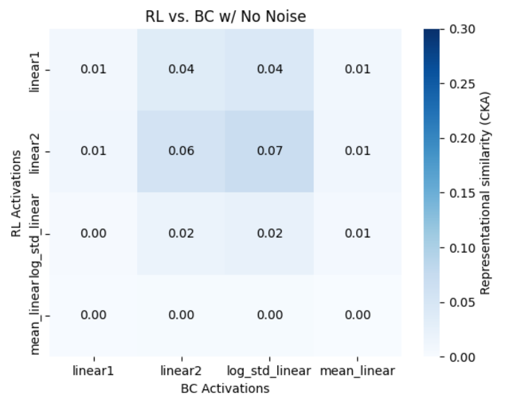
Final analysis of the kinematics of the walker agents was conducted using Power Spectral Density (PSD). PSD analyzes the frequency of time series data, which can be applied to the kinematic data generated by the Walker agents performing the walking task. In this case, the frequency of the agent's movements when walking was analyzed by PSD to detect oscillations within the frequency of the movement cycle. After the kinematic data is collected from each agent, the movement signal is sampled and then transformed with a Fourier transformation, which converts the signal from the time representation to a frequency representation. Once the data is in this form, the power per unit frequency, or PSD, can be calculated from the signal and plotted.



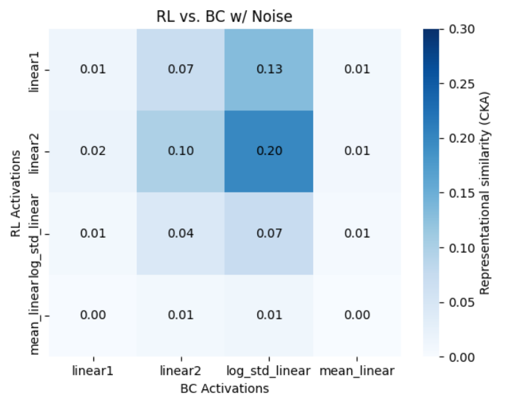
(a) CKA Between RL Network Layers



(b) CKA Between BC Network Layers



(c) CKA Between RL and BC with No Noise



(d) CKA Between RL and BC with Noise

Figure 2: CKA Similarity Between Online and Offline RL Networks

3 Results/Analysis

The results of the CKA similarity analysis are displayed in Figure 2. Plots A and B demonstrate that the activation from the online reinforcement learning and behavior cloning models are not representationally similar when comparing them to layers within their respective neural networks. If the two networks were similar, each unique combination of layers should have the same shade in both self-comparison graphs. Most notably, the log std linear layer and the linear2 layer display high similarity within the behavior cloning network; however, comparing them in the online reinforcement learning network shows very little similarity. Furthermore, plots C and D further showcase the fact that there is very little similarity between the reinforcement learning and behavior cloning models. The CKA analysis between the reinforcement learning and behavior cloning networks yields values smaller than 0.1 for all combinations of activations. This suggests that the two representations are highly dissimilar, meaning the representations and task learning significantly differ between the two agents. However, when comparing the online reinforcement learning model to the behavior cloning model with noise, it was found that they displayed higher similarity than the model without any addition of noise. This suggests that the policy learning by the behavioral cloning model with noise is more robust and more closely reflects the representations of the online reinforcement learning model from which it was trained. Contrary to what we hypothesized, it turns out that a behavior-cloning-trained agent has very different representations concerning the walking task.

Tracking the points plotted on the t-SNE plot as the agent moves about its environment, we notice the emergence of a cyclical pattern of movement within the t-SNE through time. Such a cyclical movement around the plot indicates the clear presence of a structured gait cycle, in which the agent repeats its stance in a cyclical fashion as it takes two steps in a straight line through its environment.

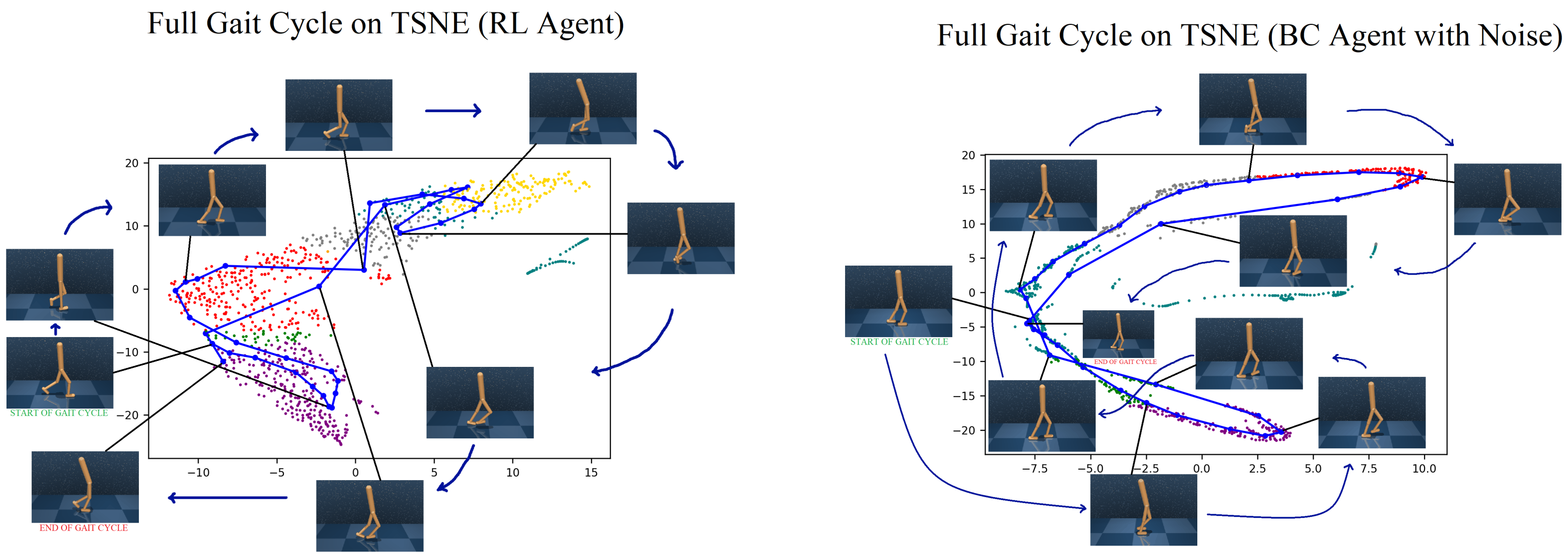


Figure 3: t-SNE plots of Gait Cycles for SAC and BC agents

In the generated t-SNE plots shown above, to which a watershed clustering has been applied (Figure 3) we see the presence of a repetitive cycle for both our SAC agent and our BC agent trained with noise, that navigates through each identified cluster before returning to its starting cluster. From this pattern, we see that as the agents move about their environments, their patterns of walking become more structured and monotonous, resulting in clear cyclical patterns appearing in the t-SNE plot.

Furthermore, by comparing the two plots, we can also see that there is less variance in the gait cycle of the BC agent (the cyclical movement within the t-SNE follows a more direct path), which is what we identify as a more consistent stride in the agent's walking movement.

Both the online and offline reinforcement learning models generate actions guided by policies that control behavior. It is important to ensure that the power spectral densities of the kinematics are preserved to ensure the learned behaviors are the same. If the PSD is not preserved between the two agents, the model may not be able to capture crucial aspects of the behavior that we want to compare. In this particular case, we further analyzed if the walking behavior is what we expect from a walking motion and is not a different action. To verify this, the behavior cloning model should be able to replicate the frequency distribution of the agent's movements. From the power spectral density analysis, it can be concluded that the harmonics present in the original online reinforcement learning model are preserved and also displayed in both the behavior cloning models with and without noise. All three models being compared, BC with noise, BC with no noise, and RL, all display highly similar movement cycles as shown by Figure 4. From the PSD analysis, we can conclude that the harmonics of the movement is preserved between online and offline reinforcement learning models.

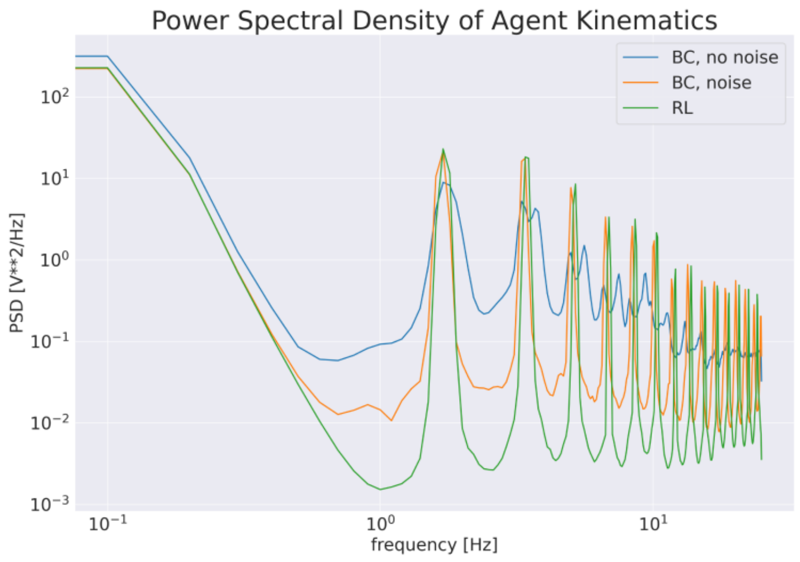


Figure 4: Power Spectral Density of Kinematics

4 Conclusion

In conclusion, the online and offline reinforcement learning models appear to be display highly dissimilar representations, as suggested by the CKA analysis. This analysis revealed close to no correlation between the representations of each model, suggesting they learned different policies for the same task. Despite being trained on the data generated from the online reinforcement learning model, the offline model ended up formulating significantly different representations and policies.

Further differences in the behavior of the two agents are exemplified through the analysis of the t-SNE plot for each agent, from which it was found that the stride of the BC agent was more strict, compared to that of the SAC agent.

This analysis provides insights of the underlying nuances within offline and online reinforcement learning algorithms. By evaluating the similarity of the representations between the two networks and the agents' behavioral patterns, we gained insights of the strengths and weaknesses of the different approaches. Overall, the findings of this study contributes to the research in the intersecting field of neuroscience and deep learning and provides footwork for further efforts of the relationship between these two variants of reinforcement learning.

References

Josh Merel, Diego Aldarondo, Jesse Marshall, Yuval Tassa, Greg Wayne, and Bence Ölveczky. Deep neuroethology of a virtual rodent, 2019. URL `https://arxiv.org/abs/1911.09451`.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth, 2020. URL `https://arxiv.org/abs/2010.15327`.

Honglak Lee Geoffrey Hinton Simon Kornblith, Mohammad Norouzi. Similarity of neural network representations revisited. *ICML 2019*, 2019. URL `https://arxiv.org/abs/1905.00414`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`.