PyTorch implementation of Deep Fluids: A Generative Network for Parameterized Fluid Simulations

VLADIMIR IVANOV, Stanford University

This project implements the the Deep Fluids [Kim et al. 2018] paper in PyTorch [Paszke et al. 2019]. The original paper presents a novel approach to generating fluid simulation velocity fields. This project studies experiments introduced in the Deep Fluids paper. The first experiment consists of training a generator network to produce a velocity field based on external parameters of a scene. Secondly, an autoencoder model for velocity fields is studied. The third experiment implements a time integration network.

In this project, emphasis is made on reproducibility. The code is open sourced and we study the effect of model architecture and hyperparameters on the learning curves and model performance. We also present a tutorial on visualizing the results in Blender.

CCS Concepts: • **Computer systems organization** → Robotics.

Additional Key Words and Phrases: Rendering, Computer Graphics

ACM Reference Format:
Vladimir Ivanov. 2020. PyTorch implementation of Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *ACM Trans. Graph.* 1, 1 (November 2020), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Over the course of several years we have seen applications of neural networks to various fields including imaging, rendering and physical simulations. The neural networks are employed in numerous capacities. In some applications, they are used to replace a single module of complex system. One such example would be [Hu et al. 2018] paper with a neural network mimicking artist's interaction with commercial software.

In other applications, a neural network is used to model a phenomenon at large. Generative Adversarial Networks are used in this fashion. For instance, [**?**] offer a different perspective on style notion in image generation. In their work, a neural network is used to produce an image end to end. The Deep Fluids paper follows the route by generating velocity fields from a latent code representation.

The development of deep learning algorithms was noticed in the physical simulation community. And in the past couple years there has been a plethora of papers on applying the algorithms to physical simulations. The milestone work [Mnih et al. 2013] showed that learning algorithms can work in simple simulated environments. With the introduction of [Schulman et al. 2017] more complex problems were being solved. This resulted in more sophisticated simulated environments being solved. Deep learning was applied to simulated environments in [Bergamin et al. 2019]. And the Deep Fluids paper brings the power of deep learning to smoke and liquid simulations at full scale.

The source code is published at https://github.com/vivanov879/deep-fluids.

Author's address: Vladimir IvanovStanford University, vivan@stanford.edu.

0730-0301/2020/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 STANDARD SIMULATION TECHNIQUES

2.1 Overview

Fluids are commonly simulated using commercial software. The simulations are used in the film and gaming industries. In this project, Mantaflow package is used for simulating liquids and smoke, see Figure 1. The package has a seamless integration with numpy [van der Walt et al. 2011] and offers tutorials in tensorflow [Abadi et al. 2016].

Fig. 1. Mantaflow simulation

2.2 Obtain Training Data

To generate the training data, a number of scenes from the Deep Fluids paper are used. A scene in Mantaflow describes sources,

solving algorithms, precision, and number of simulation steps. A mantaflow program produces both smoke density in time as well as velocity fields.

2.3 Visualization

We also use mantaflow to advect the velocity fields to obtain frames in OpenVDB [Museth et al. 2013] format. Each frame stores smoke density. Which in turn can be visualized in software tools like Maya and Blender. Mantaflow is very well integrated with the Blender package. The visualizations of experiments reproduced in this project are presented in the video playlist.

3 SCENES

In this project, we study two scenes. One is a scene with varying smoke velocity at the source of the smoke and buoyancy. The second one involves a randomly moving smoke source.

3.1 Varying Properties

The first scene consists of a source with varying smoke inflow speed and buoyancy. The training data is comprised of simulations at various levels of speed and buoyancy. We store the velocity field at each point in time. Thus, the model input is 3-dimensional: speed, buoyancy and time. While the output is the velocity field, which dimensions are 3 x Depth x Height x Width.

3.1.1 Model architecture. For this scene, we use a Generator network which consists of multiple convolutional layers interchanged with occasional upscaling operations. We use 16 convolutional layers adding up to 2 million network parameters.

3.2 Randomly moving smoke source

For the second scene, we randomly generate 200 simulations of 400 frames each. In each simulation, a smoke source is randomly moving while outputting smoke. During the experiment, we store the velocity field at each point in time as well as the source coordinates. We train an autoencoder that aims to provide a concise representation in latent space code.

We also train a time integrating network that operates over the latent space code. The network allows to produce a simulation of a moving smoke source given the initial velocity field and the positions of the moving source in time. We also study effect of using Variational Autoencoder on the quality of the velocity field predictions.

4 METHODS

4.1 Autoencoder

The Autoencoder model consists of two models: Encoder and Generator. The generator used in the Autoencoder mimicks the standalone Generator described in the first scene. However, it operates over the latent space code with is of size 16 in our second scene. The Encoder resemebles the Generator network with no upscaling operations. It is designed to be of similar capacity like the Generator, thus it consists of 2 million parameters.

Apart from the reconstruction loss, the autoencoder makes sure the last of its encoded values are equal to the positions of the smoke source at each point in time. This allows to create a different learning signal for the Autoencoder.

This project also introduces an option of using a Variational Autoencder which places even more regularization on the latent space code.

4.2 Time Integrating Neural Network

The time integrating neural network is the most lightweight network in the project. The reason is it works over the latent space with is of size 16. It is quick to train, however the results produces by the network are least consistent. This is due to the fact that at inference time we use the Neural Network recursively on its own outputs which results in accumulating error. To counter that, we train network over periods of 30 frames: the network aims not only to predict the next frame, but the next 30 frames as well. This allows to add more consistency. However, in our experiments, we have time spans of 400 frames, and the behavior doesn't look that realistic on the last simulated frames compared to the first ones.

5 RESULTS

5.1 Generator in the scene of varying external parameters

In this experiment, we want to overfit our dataset with a neural network. Since we want to query the latent code for velocity fields. As we have two million in our network, this is an easy task. The learning curves look solid at Figure 2

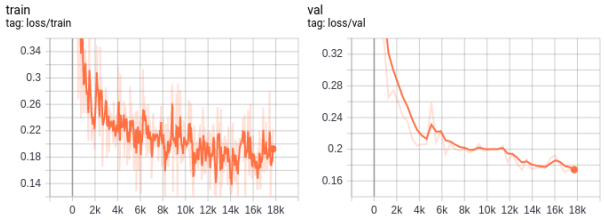

Fig. 2. Train and validation learning curves for the first scene. Generator network

The visualization of the latent code at inference time looks plausible. See Figure 3

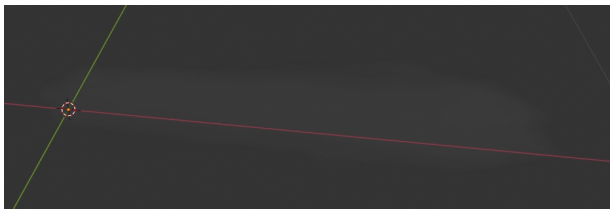

Fig. 3. Visualization of the generated smoke density in Blender

5.2 Autoencoder in the moving source scene

Since we are training both encoder and decoder, this is the most resource consuming experiment. Yet, the learning curves are stable and the training is well behaved. For this experiment, the dataset is randomly split into train and validation. The learning curves can be observed at Figure 4.

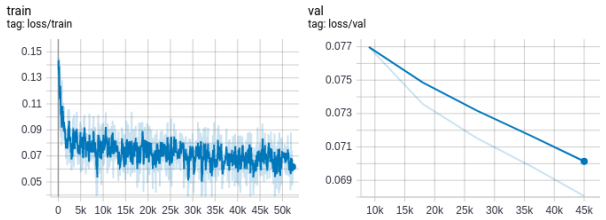

Fig. 4. Visualization of the generated smoke density in Blender

5.3 Autoencoder for velocity field reconstruction

To assess the efficiency of the autoencoder, we use the Autoencoder's Generator network onto the latent space code produced by the Autoencoder's Encoder. The Blender visualizations look natural. Refer to Figure 5

5.4 Time Integrating Neural Network

This is the most light weight model. So, the experiment iteration is quick. However, since we work with latent code and our implicit aim is to produce quality velocity fields, this is a hard network to train. We tried a number of network activations and have not achieved a perfect match. This can be explained by the fact that at inference time we recursively call the network on its own output. While in training we only use a window of 30 frames.

To avoid the situations when values go to infinity during inference over 400 frames, I tried adding a Tanh activation at the network last layer. Adding the Tanh activation does not allow the network to achieve a perfect score on either train or validation set, see Figure 6 The validation set consists of simulations not present in the training set.

This can be explained by the fact that when ideally matching the output at 30 frames window during training, the network doesn't see imperfect predictions when training. And the probability of seeing imperfect inputs is high during inference, since we did not train for the 400 window seen at inference.

When using a Variational Autoencoder instead of ordinary Autoencoder at previous step, the learning curves do not go 0 loss both for Tanh and no Tanh cases. Refer to Figure 7

5.5 Effect of Variational Autoencoder on velocity fields

The Time Itegrating Neural Netowrk when paired with Variational Autoencoder does not produce well behaved velocity fields. See Figure 8 and Figure 9. However due to the noise imposed by Variational part when generating data for the Time Integrating Neural Network, the velocity field values do not explode when running inference for 400 steps. As described above, the 400-step inference results

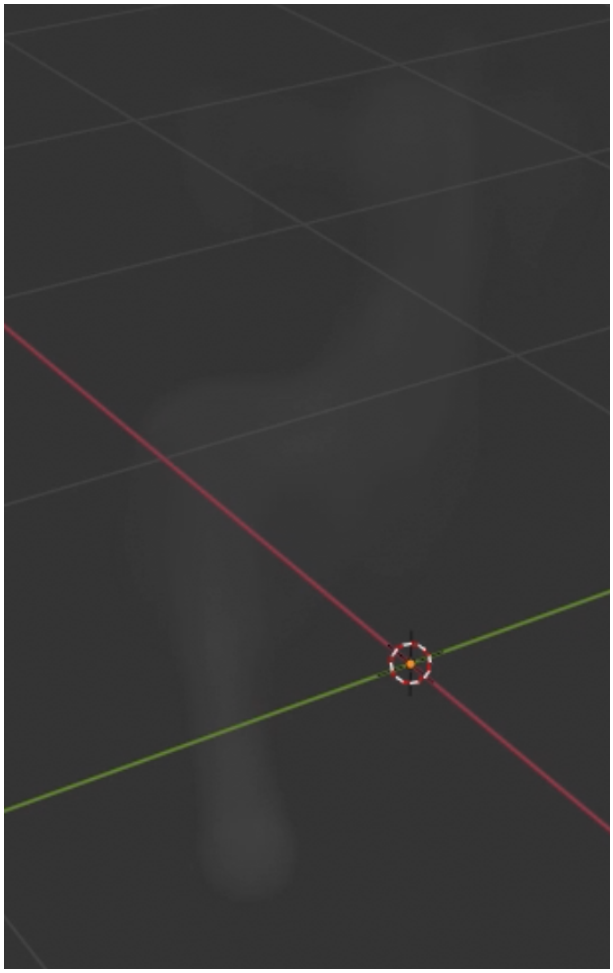

Fig. 5. Visualization of the ground truth smoke density in Blender

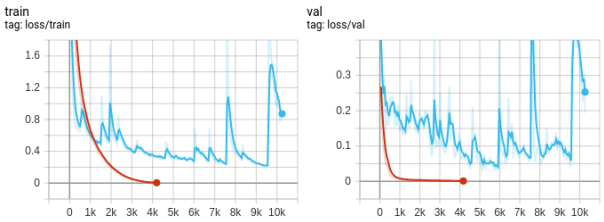

Fig. 6. The learning curves do not end up at 0 with Tanh activation. But the resulting velocity fields look better with Tanh. Blue line: Tanh activation. Red line: no Tanh activation

in imperfect inputs. And the noisy inputs when using Variational Autoencder prepare the Time Integrating Network for the situation. Since the velocity field values are finite, the smoke doesn't entirely disappear from the screen. For the future work, it is interesting to investigate the effect of the Variational Autoencoder network

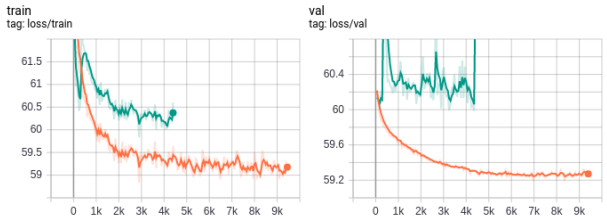

Fig. 7. The learning curves for the Time Integrating Neural Network when used in pair with a Variational Autoencder. The green line is for the case when Tanh activation is used. The orange one depicts no Tanh use.

architecture and latent space dimensionality on the quality of the velocity fields.

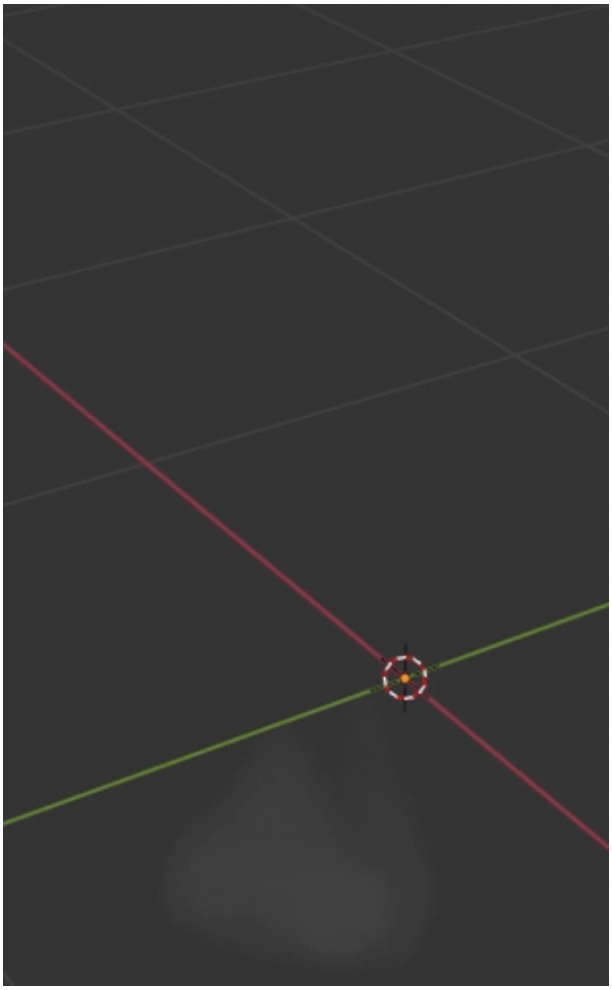

Fig. 8. Blender visualization for Variational Autoencoder with Tanh at last layer of the time integrating neural network. The smoke doesn't raise and stays near the source all the time.

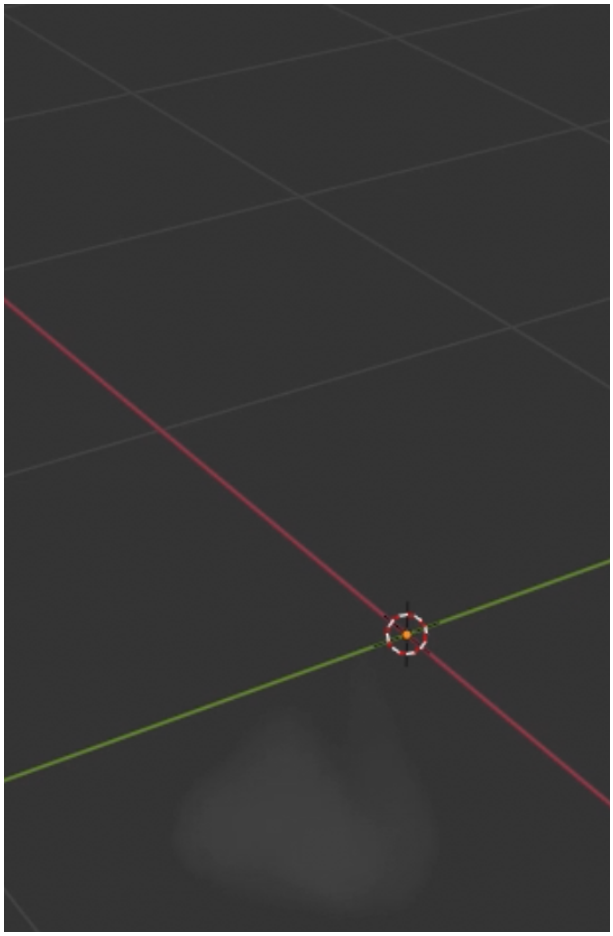

Fig. 9. Blender visualization for Variational Autoencoder with no Tanh at last layer of the time integrating neural network. The smoke is located near the source till the end of the simulation.

5.6 The effect of Tanh activation when time integrating network is used in pair with an ordinary Autoencoder

The Tanh activation produces a better visualization even though the loss is increased with Tanh. Contrast the Figure 11 and Figure 10.

5.7 Training a Variational Autoencoder

When training a Variational Autencoder, the learning curves achieve plateau after a single training epoch. Refer to Figure 12. This can be explained by the supposedly better behaved latent space code induced by the Variational part of the Autoencoder.

6 FUTURE WORK

In addition to the Variational Encoder benchmarking described above, it is worth investigating using an LSTM [Sundermeyer et al. 2012] network to better handle the exploding velocity field values in the Time Integrating Neural Network. Also, a larger window can be tried at the expense of correlated training inputs.

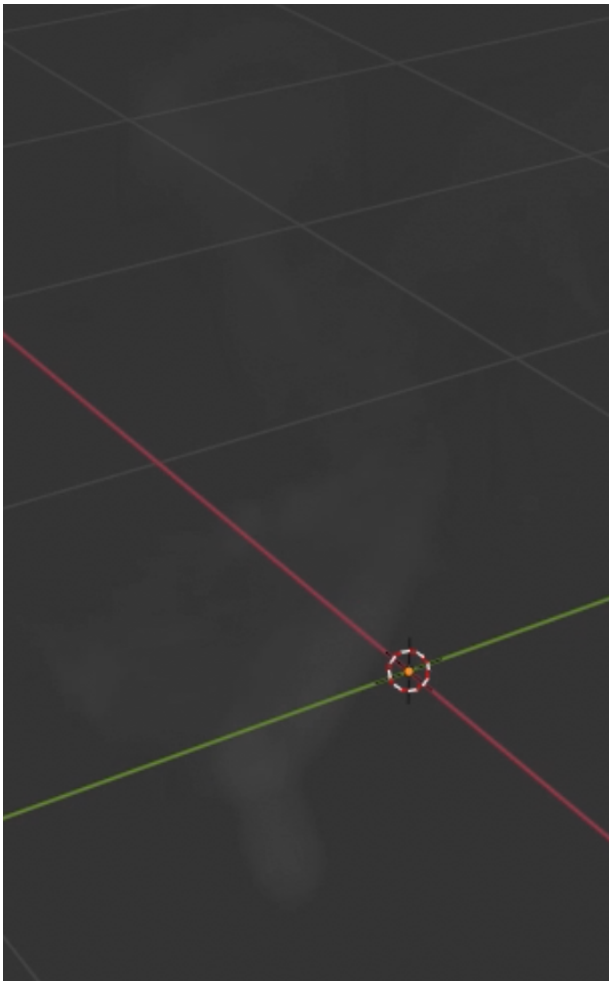

Fig. 10. Blender visualization for Autoencoder with no Tanh at last layer of the time integrating neural network. The inference runs fine for the first 200 frames. But feeding the network output into its own input recursively results in nearly infinity values after 200 frames. As a result, the smoke is dispersed in all directions near 200-th frame, and doesn't on the screen any more.

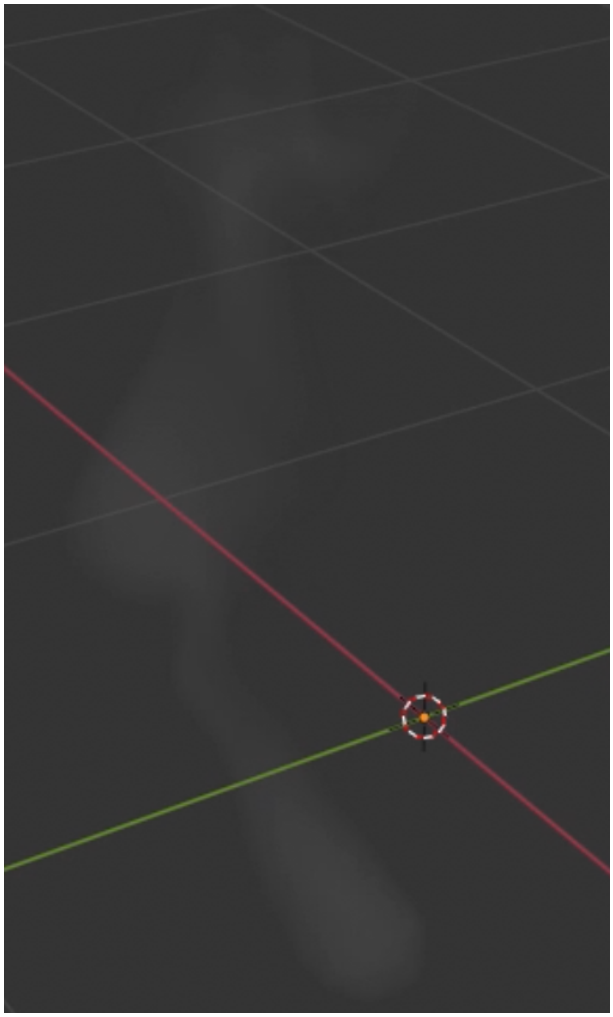

Fig. 11. Blender visualization for Autoencoder with Tanh at last layer of the time integrating neural network. The simulation is consistent and very much resembles ground truth simulation.

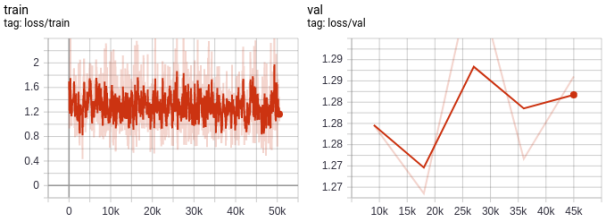

Fig. 12. The learning curves for training Variational Autoencoder

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. 2016. TensorFlow: A system for large-scale machine learning. *CoRR* abs/1605.08695 (2016). arXiv:1605.08695 http://arxiv.org/abs/1605.08695

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: data-driven responsive control of physics-based characters. *ACM Trans. Graph.* 38, 6 (2019), 206:1–206:11. https://doi.org/10.1145/3355089.3356536

Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. 2018. Exposure: A White-Box Photo Post-Processing Framework. *ACM Trans. Graph.* 37, 2 (2018), 26:1–26:17. https://doi.org/10.1145/3181974

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus H. Gross, and Barbara Solenthaler. 2018. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *CoRR* abs/1806.02071 (2018). arXiv:1806.02071 http://arxiv.org/abs/1806.02071

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Aldén, Peter Cucka, David R. Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2013, Anaheim, CA, USA, July 21-25, 2013, Courses*. ACM, 19:1. https://doi.org/10.1145/2504435.2504454

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM Neural Networks for Language Modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*. ISCA, 194–197. http://www.isca-speech.org/archive/interspeech_2012/i12_0194.html

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *CoRR* abs/1102.1523 (2011). arXiv:1102.1523 http://arxiv.org/abs/1102.1523