Category: **Physical Sciences**

GOOGLE DEEPMIND RESEARCH:

Learning to Simulate Complex Physics with Graph Networks

Tran Tue Nhi

TABLE OF CONTENTS

01

PROJECT OVERVIEW

RESEARCH BACKGROUND

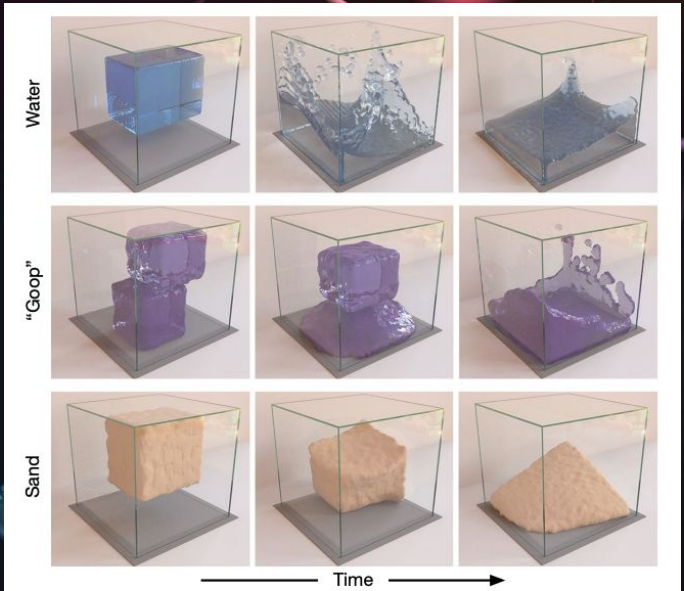


Figure 1. Rollouts of our GNS model for our WATER-3D, GOOP-3D and SAND-3D datasets. It learns to simulate rich materials at resolutions sufficient for high-quality rendering [video].

- **History:** "Learning to Simulate Complex Physics with Graph Networks" is a research paper published by DeepMind in 2021. In this paper, the authors present a new approach to physics simulation using graph networks.
- **Problem Statement:** The authors aim to address the limitations of traditional physics simulators, which are based on differential equations, in handling complex physical systems. The traditional physics simulators are computationally expensive and require expert knowledge to set up and maintain. Additionally, the traditional simulators are limited in their ability to handle complex relationships between physical entities.
- **Objectives:**
  - To demonstrate the effectiveness of graph networks in representing and processing complex relationships between physical entities
  - To present a new approach to physics simulation using graph networks
  - To propose a new graph network architecture, called a Physics Graph Network (PGN), for simulating physical systems
  - To show that PGNs can be trained end-to-end to minimize the difference between observed interactions and the predictions of the model
  - To demonstrate the ability of PGNs to learn to simulate physical systems such as particle systems, cloth, and fluid dynamics.

Reference: Learning to Simulate Complex Physics with Graph Networks. G. Kaelbling, R. Boukhelifa, K. Dey, A. Dunning, A. Li, F. Yang, J. B. Tenenbaum, and T. L. Griffiths. arXiv preprint arXiv:2102.06350, 2021.

PROJECT SUMMARY

1. PROJECT REQUIREMENTS

4. **Graph Nerual Network for Fluid Flow So,l;atopm:**
   a. This is an especially interesting project that is the most data-driven of the lot. Basically, the goal is to first re-produce this paper by Deep Mind by Gonzalez et al. (with co-authors from DeepMind).
   b. Even re-creating the pipeline of this paper (with all datasets etc. ready) will be a worthwhile task for a 3 month project to do it correctly and cleanly.
   c. **Goal**: Ideally the goal is to create a better GNN model for fluid flow learning.

2. PROJECT PROGRESS

Reproduced the paper of DeepMind → Created a better GNN model for fluid flow learning

Reference: VinUni Instructure

02

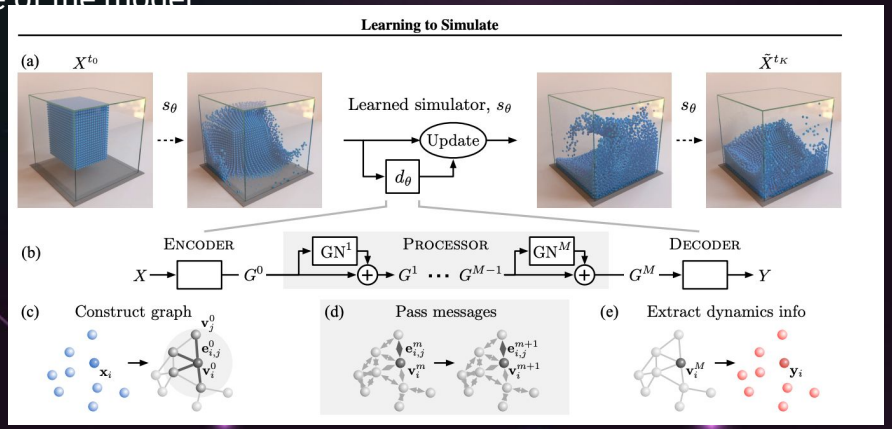
REPRODUCTION

REPRODUCTION PROCESS

Reference: Learning to Simulate Complex Physics with Graph Networks. G. Kaelbling, R. Boukhelifa, K. Der, J. Dunning, A. H. F. Yang, J. B. Tenenbaum, and T. L. Griffiths. arXiv preprint arXiv:2102.06359, 2021.

DATA PROCESSING

- The research uses **real-world observations of physical interactions to train their Physics Graph Network (PGN) architecture**. The data used in this research was collected using a high-speed camera system that captures the position, velocity, and other physical properties of particles, cloth, or fluids at high temporal and spatial resolution.
- The collected data was then **preprocessed to convert it into a format suitable for input into the PGN**. The authors use a graph representation of the physical system, where nodes in the graph represent physical entities and edges represent physical interactions between those entities. The data for each node includes the position, velocity, and other physical properties of the entity.
- The preprocessed data was then **split into training, validation, and testing sets.** The training set was used to train the PGN, while the validation set was used to tune the hyperparameters of the model, and the testing set was used to evaluate the performance of the model.

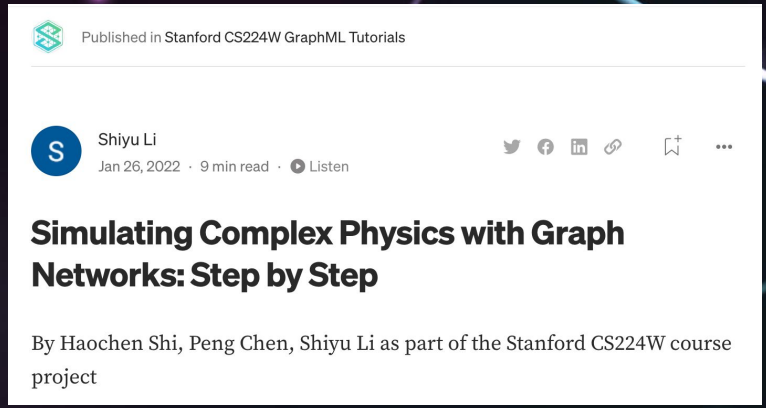


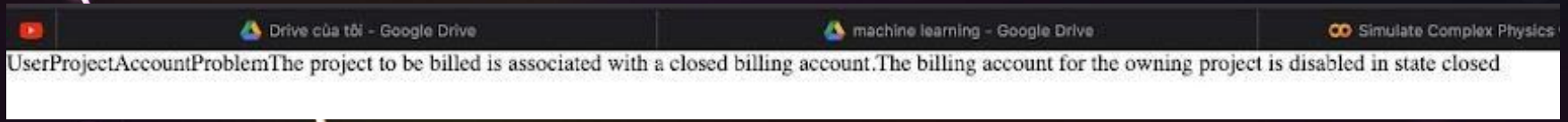
Reference: Learning to Simulate Complex Physics with Graph Networks. G. Kaelbling, R. Boukhelifa, K. Der, J. Dunning, A. H. F. Yang, J. B. Tenenbaum, and T. L. Griffiths. arXiv preprint arXiv:2102.06359, 2021.

EXPERIMENT 1

- **Approach:** Experimented with WaterDrop dataset of Stanford CS224W Graph Machine Learning to simulate complex physics with graph networks in Gooogle Colaboratory

- **Challenge 1:** When I trained the WaterDrop dataset, it had the problem that the dataset was closed due to the fact that the billing account of owner was disabled.



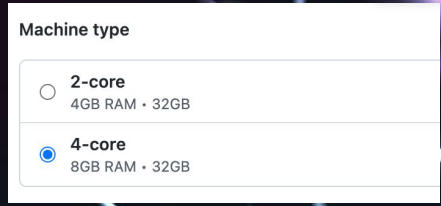


- **Solution:** I decided to use the Paper and GitHub of DeepMind as the main source of reference for reproducing the simulation of complex physics with graph networks.

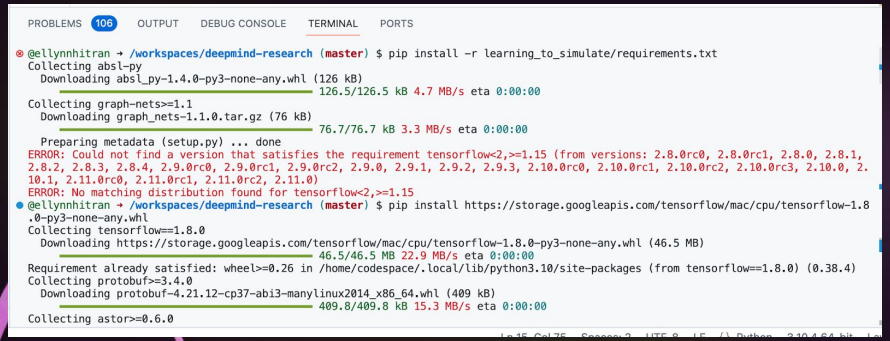
Reproduction

Experiment 2 – Step 1

- **- Methodology:** I fully reproduced the research by using Cloud with the setting of 32GB with 8GB RAM and 4-core.



A. Step 1: Install package requirements



- **- Challenge:** In the process of downloading requirements, I came across a problem of downloading Tensorflow version 1.15+

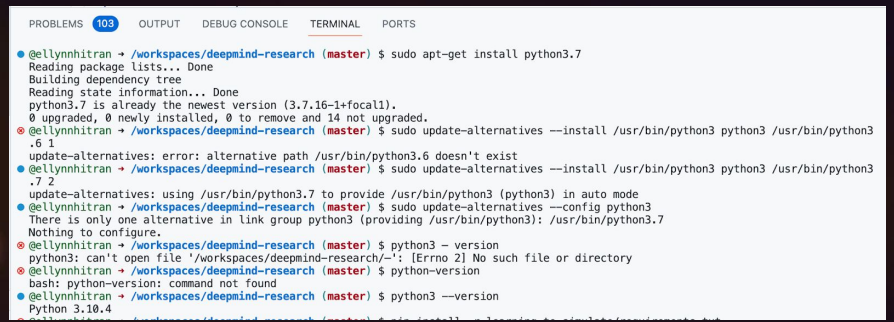
```
10.1, 2.11.0rc0, 2.11.0rc1, 2.11.0rc2, 2.11.0)
ERROR: No matching distribution found for tensorflow<2,>=1.15
```

=> It turns out that my current Python version is 3.10.4, which requires Tensorflow 2.2 or later. Therefore, I got trouble while trying to install the Tensorflow version that is larger than 1.15 and smaller than 2.0.

- **- Solution 1:** For this problem, I can approach 2 ways:
  + Change the Tensorflow of the system
  + Or downgrade my Python version.

If I change the source code to use Tensorflow 2.2, it will also change the system.

But if I directly downgrade my current Python version to 3.7, it did not work out at all.

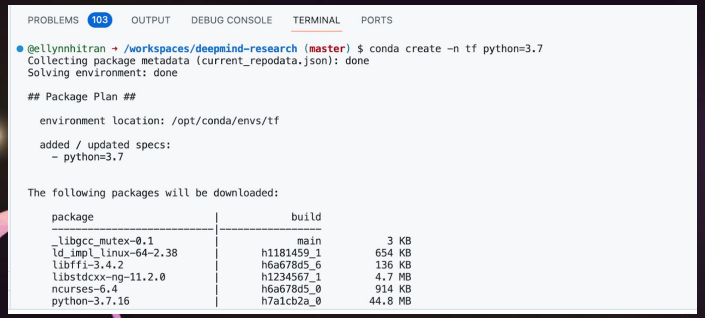


No way to go?

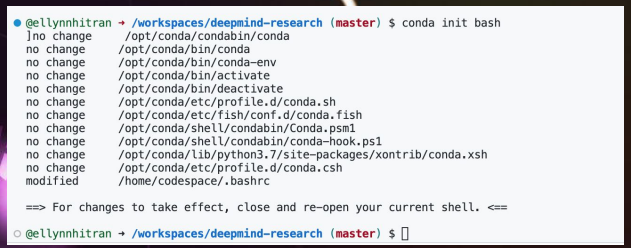
Experiment 2 – Step 1

- **- Solution 2:** When searching for a solution, I found out that I could a virtual environment where my Python version was 3.7 so I would be able to download the Tensorflow 1.5

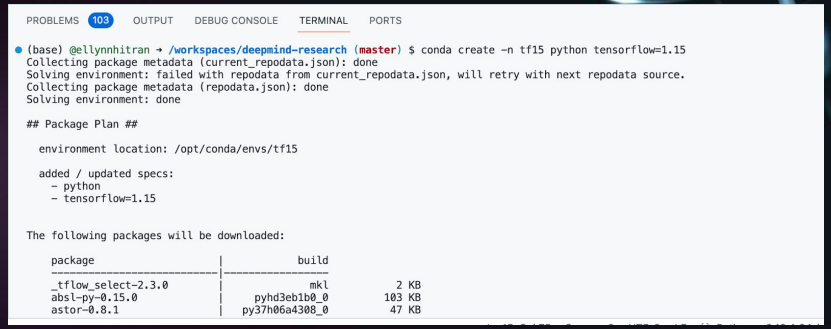
1. Set up a virtual environment with the required version Python 3.7



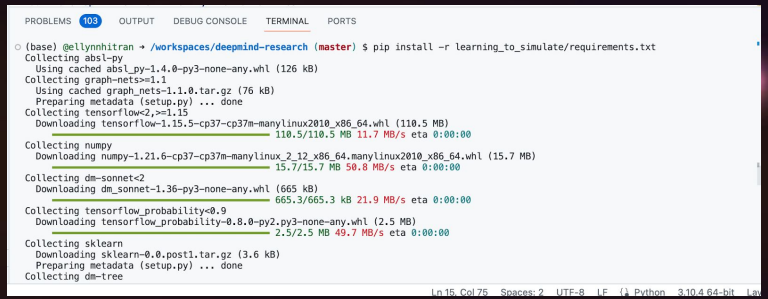
2. Initialize bash shell of new environment



3. Install tensorflow version 1.15



4. Successfully install package requirement of research



Experiment 2 – Step 2

- **B. Step 2: Access and download the dataset**

```
PROBLEMS 106  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● @ellynnhitran → /workspaces/deepmind-research (master) $ mkdir -p /tmp/datasets
● @ellynnhitran → /workspaces/deepmind-research (master) $ bash ./learning_to_simulate/download_dataset.sh WaterRamps /tmp/datasets
--2023-01-30 05:30:27--  https://storage.googleapis.com/learning-to-simulate-complex-physics/Datasets/WaterRamps/metadata.json
Resolving storage.googleapis.com (storage.googleapis.com)... 172.253.62.128, 142.251.163.128, 142.251.16.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.253.62.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 366 [application/octet-stream]
Saving to: '/tmp/datasets/WaterRamps/metadata.json'

/tmp/datasets/WaterRamps/metadata 100%[=============================================================>]     366  --.-KB/s    in 0s

2023-01-30 05:30:28 (46.5 MB/s) - '/tmp/datasets/WaterRamps/metadata.json' saved [366/366]

--2023-01-30 05:30:28--  https://storage.googleapis.com/learning-to-simulate-complex-physics/Datasets/WaterRamps/train.tfrecord
Resolving storage.googleapis.com (storage.googleapis.com)... 172.253.62.128, 142.251.163.128, 142.251.16.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.253.62.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7350243340 (6.8G) [application/octet-stream]
Saving to: '/tmp/datasets/WaterRamps/train.tfrecord'

/tmp/datasets/WaterRamps/train.tf 100%[=============================================================>]   6.84G   169MB/s    in 47s

Ln 15, Col 75  Spaces: 2  UTF-8  LF  {} Python  3.10.4 64-bit  Layou
```

- **Challenge 1:** I got problem “Descriptors cannot not be created directly” when running the simulation of model. This problem happened because the generated code is out of date and must be regenerated with protobuf >= 3.19.0.

```
PROBLEMS 103  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● (base) @ellynnhitran → /workspaces/deepmind-research (master) $ mkdir -p /tmp/models
⊗ (base) @ellynnhitran → /workspaces/deepmind-research (master) $ python -m learning_to_simulate.train \
>     --data_path=/tmp/datasets/WaterRamps \
>     --model_path=/tmp/models/WaterRamps
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/opt/conda/lib/python3.7/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/workspaces/deepmind-research/learning_to_simulate/train.py", line 41, in <module>
    import tensorflow.compat.v1 as tf
  File "/opt/conda/lib/python3.7/site-packages/tensorflow/__init__.py", line 102, in <module>
    from tensorflow_core import *
  File "/opt/conda/lib/python3.7/site-packages/tensorflow_core/__init__.py", line 28, in <module>
    from tensorflow.python import pywrap_tensorflow  # pylint: disable=unused-import
  File "<frozen importlib._bootstrap>", line 1019, in _handle_fromlist
  File "/opt/conda/lib/python3.7/site-packages/tensorflow/__init__.py", line 50, in __getattr__
    module = self._load()
  File "/opt/conda/lib/python3.7/site-packages/tensorflow/__init__.py", line 44, in _load
    module = _importlib.import_module(self.__name__)
  File "/opt/conda/lib/python3.7/importlib/__init__.py", line 127, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
```

- **Solution:** I downgraded the protobuf to be 3.20

```
PROBLEMS 103  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● (base) @ellynnhitran → /workspaces/deepmind-research (master) $ pip install protobuf==3.20.*
Collecting protobuf==3.20.*
  Downloading protobuf-3.20.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 24.2 MB/s eta 0:00:00
Installing collected packages: protobuf
  Attempting uninstall: protobuf
    Found existing installation: protobuf 4.21.12
    Uninstalling protobuf-4.21.12:
      Successfully uninstalled protobuf-4.21.12
Successfully installed protobuf-3.20.3
⊗ (base) @ellynnhitran → /workspaces/deepmind-research (master) $ python -m learning_to_simulate.train \
> --data_path=/tmp/datasets/WaterRamps \--model_path=/tmp/models/WaterRamps
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/opt/conda/lib/python3.7/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/workspaces/deepmind-research/learning_to_simulate/train.py", line 46, in <module>
    from learning_to_simulate import learned_simulator
  File "/workspaces/deepmind-research/learning_to_simulate/learned_simulator.py", line 31, in <module>
    from learning_to_simulate import connectivity_utils
  File "/workspaces/deepmind-research/learning_to_simulate/connectivity_utils.py", line 21, in <module>
```

- **Challenge 2:** No module named 'sklearn’
- **Solution:** Installed the scikit learn package in the virtual environment

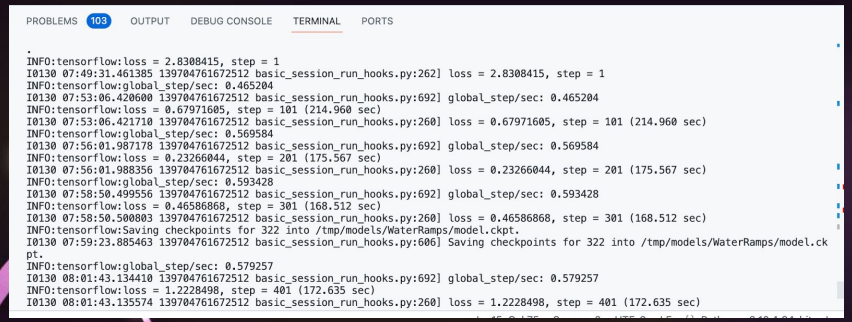
```
PROBLEMS 103  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● (base) @ellynnhitran → /workspaces/deepmind-research (master) $ pip install -U scikit-learn scipy matplotlib
Collecting scikit-learn
  Downloading scikit_learn-1.0.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (24.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.8/24.8 MB 40.9 MB/s eta 0:00:00
Collecting scipy
  Downloading scipy-1.7.3-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (38.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.1/38.1 MB 27.5 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-packages (3.5.3)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
Collecting joblib>=0.11
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 298.0/298.0 kB 13.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.6 in /opt/conda/lib/python3.7/site-packages (from scikit-learn) (1.18.5)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (4.38.0)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (23.0)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from kiwisolver>=1.0.1->matplotlib) (4.4.
```

Experiment 2 – Step 3

- **C. Step 3: Train the model**

After each training step went by, the training loss decreased (the training step interval = 100)

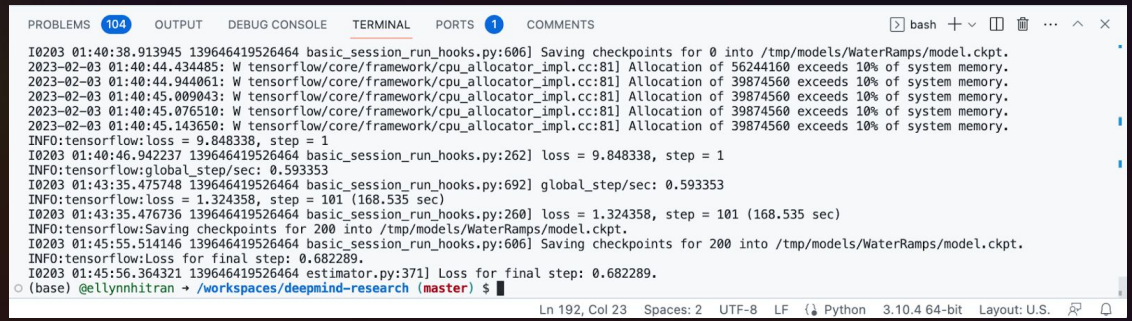


- **Challenge:** After 3 hours, the training suddenly interrupted due to the out-of-memory (in RAM consumption).

- **Why:** I found out that this was because the number of steps during training of Google Deepmind is 2e7 = 20,000,000 steps which is a very large number with the 32GB & 8GB RAM of cloud.

```
56  flags.DEFINE_string('data_path', None, help='The dataset directory.')
57  flags.DEFINE_integer('batch_size', 2, help='The batch size.')
58  flags.DEFINE_integer('num_steps', int(2e7), help='Number of steps of training.')
59  flags.DEFINE_float('noise_std', 6.7e-4, help='The std deviation of the noise.')
60  flags.DEFINE_string('model_path', None,
61                      help=('The path for saving checkpoints of the model. '
62                            'Defaults to a temporary directory.'))
```

- **Solution:** Therefore, I changed the number of steps to 2e3 (2000 steps) so it could be trained efficiently in my computer. And it worked!



Experiment 2 – Step 4

- **D. Step 4: Generated some trajectory deployments on testing dataset**

- Challenge: Allocation exceeds 10% of system memory. As the number of rollouts is high so it takes time to generate some trajectory rollouts on the test set

```
2023-02-01 01:49:21.890454: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 56244160 exceeds 10% of system memory.
2023-02-01 01:49:22.461166: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 16312320 exceeds 10% of system memory.
2023-02-01 01:49:22.498233: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 16312320 exceeds 10% of system memory.
2023-02-01 01:49:22.530986: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 16312320 exceeds 10% of system memory.
2023-02-01 01:49:22.565910: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 16312320 exceeds 10% of system memory.
```

- Multiple solutions:

- + Reduce the number of batch size from 6 to 2 to decrease number of rollouts (doesn't work).
- + Reduce sequence length size to 2 (doesn't work)
- + Change distribution of Tensorflow (doesn't work)
- + Change the GPU allocation (doesn't affect anything)

+ Limit the number of rollouts in the testing phase by using itertools

```python
# Iterate through rollouts saving them one by one.
import itertools

metadata = _read_metadata(FLAGS.data_path)
num_rollouts = 5 # change this to limit the number of rollouts
rollout_iterator = itertools.islice(rollout_estimator.predict(
    input_fn=get_input_fn(FLAGS.data_path, batch_size=1,
                          mode='rollout', split=FLAGS.eval_split)), num_rollouts)
```

- And it works!

```
INFO:tensorflow:Restoring parameters from /tmp/models/WaterRamps/model.ckpt-20
I0201 01:51:16.559202 139879221449536 saver.py:1284] Restoring parameters from /tmp/models/WaterRamps/model.ckpt-20
INFO:tensorflow:Running local_init_op.
I0201 01:51:17.030570 139879221449536 session_manager.py:500] Running local_init_op.
INFO:tensorflow:Done running local_init_op.
I0201 01:51:17.076805 139879221449536 session_manager.py:502] Done running local_init_op.
I0201 01:53:05.849027 139879221449536 train.py:477] Saving: /tmp/rollouts/WaterRamps/rollout_test_0.pkl.
I0201 01:55:01.006954 139879221449536 train.py:477] Saving: /tmp/rollouts/WaterRamps/rollout_test_1.pkl.
I0201 01:58:21.319446 139879221449536 train.py:477] Saving: /tmp/rollouts/WaterRamps/rollout_test_2.pkl.
I0201 02:00:43.645213 139879221449536 train.py:477] Saving: /tmp/rollouts/WaterRamps/rollout_test_3.pkl.
I0201 02:02:03.523889 139879221449536 train.py:477] Saving: /tmp/rollouts/WaterRamps/rollout_test_4.pkl.
```

Experiment 2 – Step 5

- **E. Step 5:** Plot trajectory of the test set

- Challenge: “Animation was deleted without rendering anything”. The error occurs because the animation is not shown

```
(base) @ellynnhitran → /workspaces/deepmind-research (master) $ python -m learning_to_simulate.render_rollout \
>     --rollout_path=/tmp/rollouts/WaterRamps/rollout_test_0.pkl
/opt/conda/lib/python3.7/site-packages/matplotlib/animation.py:888: UserWarning: Animation was deleted without rendering anything. This is most likely
 not intended. To prevent deletion, assign the Animation to a variable, e.g. `anim`, that exists until you have outputted the Animation using `plt.sho
w()` or `anim.save()`.
  'Animation was deleted without rendering anything. This is '
(base) @ellynnhitran → /workspaces/deepmind-research (master) $
```

- Solution: Add plt.ion() to turn on the interactive mode of Matplotlib and close the animation figure after display

```
# Add plt.ion() to turn on the interactive mode of matplotlib
plt.ion()

unused_animation = animation.FuncAnimation(
    fig, update,
    frames=np.arange(0, num_steps, FLAGS.step_stride), interval=10)

unused_animation.save('rollout.gif', dpi=80, fps=30, writer='imagemagick')

plt.show(block=FLAGS.block_on_show)
plt.pause(0.001)

# Add plt.close() to close the animation
plt.close(unused_animation._fig)
```

- Challenge 2: Simulation did not appear after running the simulation

```
(base) @ellynnhitran → /workspaces/deepmind-research (master) $ python -m learning_to_simulate.render_rollout \
>     --rollout_path=/tmp/rollouts/WaterRamps/rollout_test_0.pkl
```

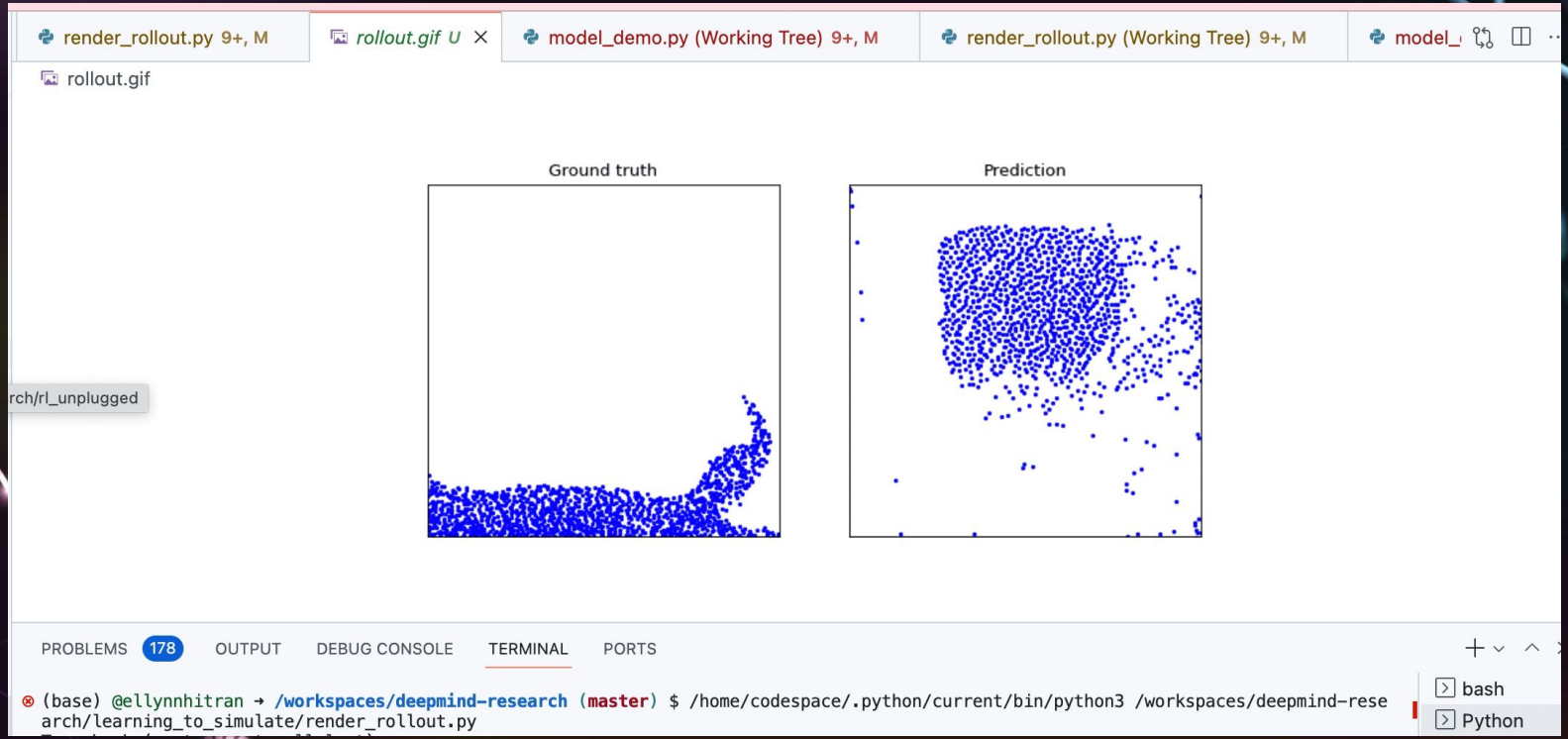
- Solution:

+ First, install interactive matplotlib framework

+ Save the simulation into a gif by using writer ‘imagemagick

```
unused_animation.save('rollout.gif', dpi=80, fps=30, writer='imagemagick')
```

Reproduction Simulation

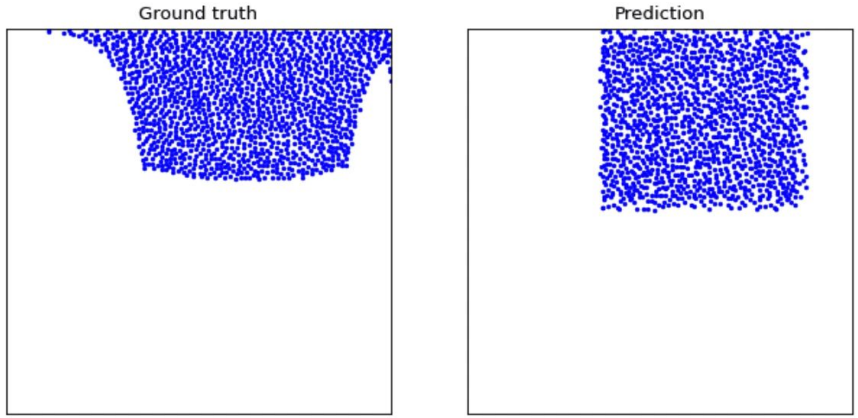


MSE = 0.443, number of steps = 2e3 (20,000 steps), step interval = 100, batch_size = 2, number of rollouts = 10

Another Simulation

[Preview] README.md | train.py 3, M | *rollout.gif U* | render_rollout.py M | model_demo.py (Working Tree) M | rende

rollout.gif



PROBLEMS 3 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | Python

03

NOVELTY

NOVELTY #1: OPTIMIZERS SELECTION

1. WHY NEEDS TO CHANGE?

- Currently, DeepMind is using Adam Optimizer as the main methodology to optimize its model parameters

- Problem: However, according to Towards AI, Adam Optimizer should not be the default learning algorithms as Adam has weak empirical generalization capability (though fast in convergence).

- Article published in September 2019, *"Bounded Scheduling Method for Adaptive Gradient Methods"* investigates the factors that lead to poor performance of Adam while training complex neural networks.
  - + ***The non-uniform scaling of the gradients*** will lead to the poor generalization performance of adaptive gradients methods.
  - + ***The exponential moving average*** used in Adam can't make the learning rate monotonously decline, which will cause it to fail to converge to an optimal solution and arise the poor generalization performance.
  - + ***The learning rate*** learned by Adam may ***circumstantially be too small*** for effective convergence, which will make it fail to find the right path and converge to a suboptimal point.

2. METHODOLOGIES

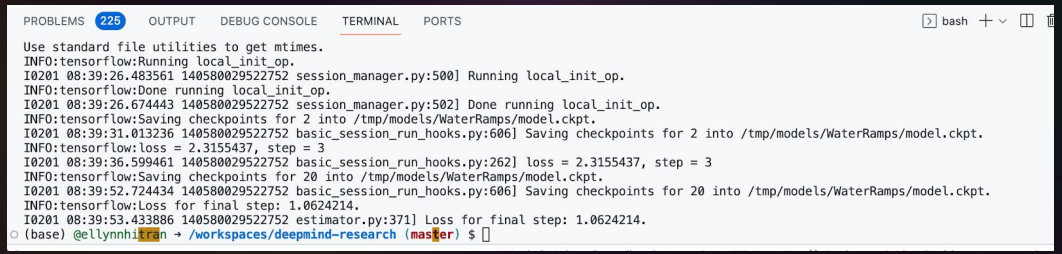
- Determine which optimizers I would use for the model: Stochastic Gradient Descent (SGD), Adadelta Optimizer and RMSROP optimizer (Sanket, 2019)
- Train the model multiple times with targeted optimizers.
- Evaluate the performance of the model

NOVELTY #1: OPTIMIZERS SELECTION

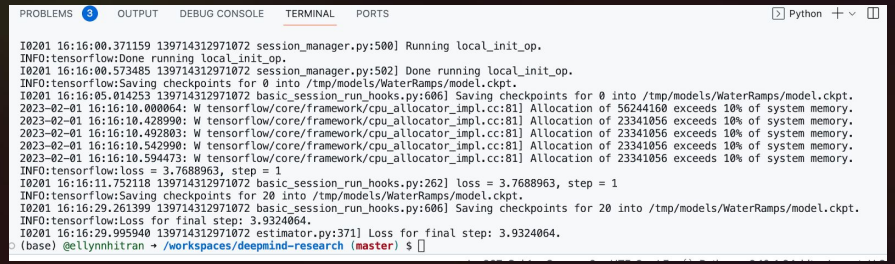
3. RESULTS

- Settings: number of steps=2e2, latent_size=128, hidden_size=128, hidden_layers=2, learning_rate=1e-4 - min_lr (1e-6), batch_size = 1

- Adam Optimizer: MSE = 1.06 (base case)



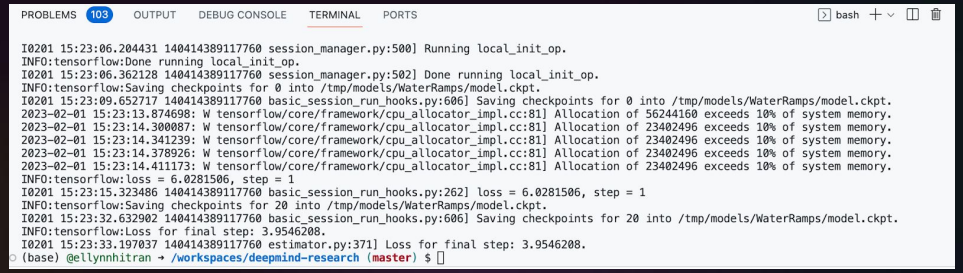
- Adadelta Optimizer: MSE = 3.924



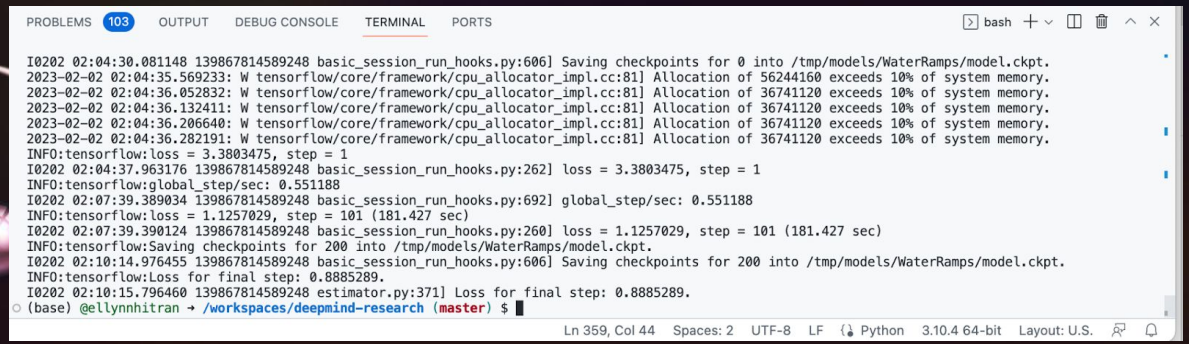
NOVELTY #1: OPTIMIZERS SELECTION

3. RESULTS

- SGD Optimizer: MSE = 3.945

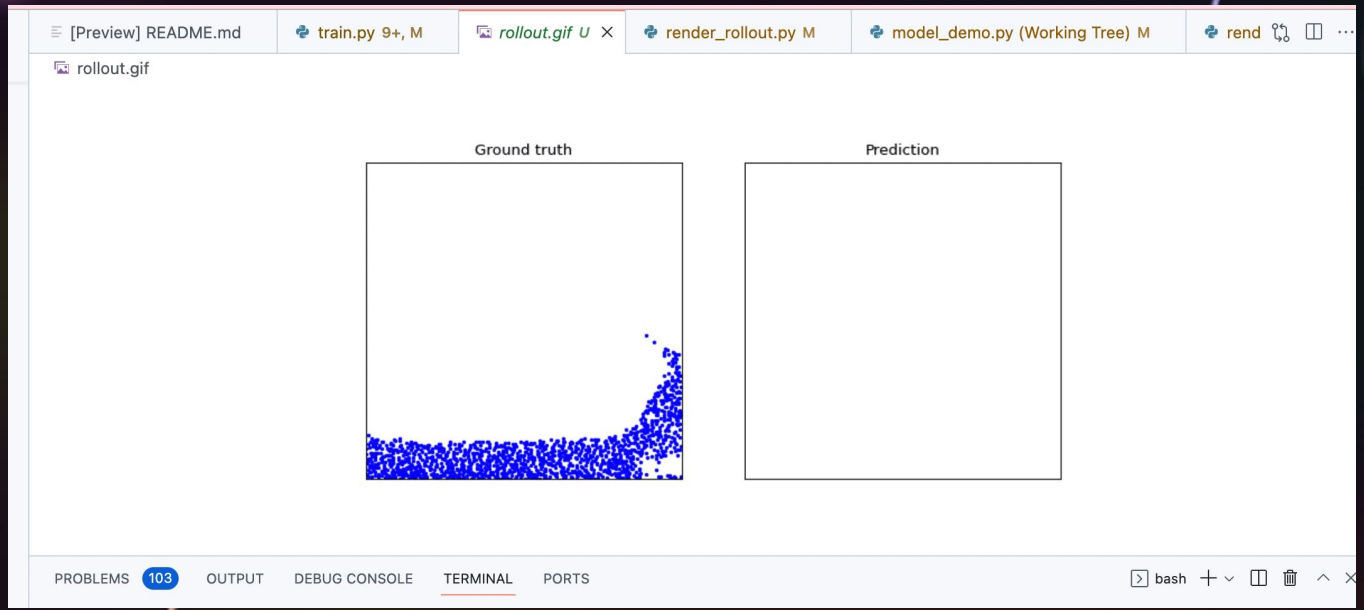


- RMSProp Optimizer: MSE = 0.885 (less than 20% in comparison with Adam Optimizer)



NOVELTY #1: OPTIMIZERS SELECTION

4. Simulate the best optimizer



Evaluation: Though RMSProp Optimizer has the smallest MSE, its accuracy might not be as good as Adam Optimizer due to the fact that the current setting of the model is for Adam. However, RMSProp has the potential to improve the model in terms of minimizing loss with the condition that other factors in the model should also adjust too.

NOVELTY #2: HYPERPARAMETER TUNING

1. WHY NEEDS TO CHANGE?

- Problem: Battaglia et al in "Relational inductive biases, deep learning, and graph networks" indicate that there is no single set of hyperparameters that works well across all problems, and suggest using a combination of grid search, random search, and expert intuition to determine the optimal values.

-The authors in "A comprehensive survey on graph neural networks" (Wu et al., 2020) suggest that hyperparameter tuning is often a crucial step in getting good performance from graph neural networks, and that grid search, random search, or Bayesian optimization can be used to find the best hyperparameters for a given task.

2. METHODOLOGIES

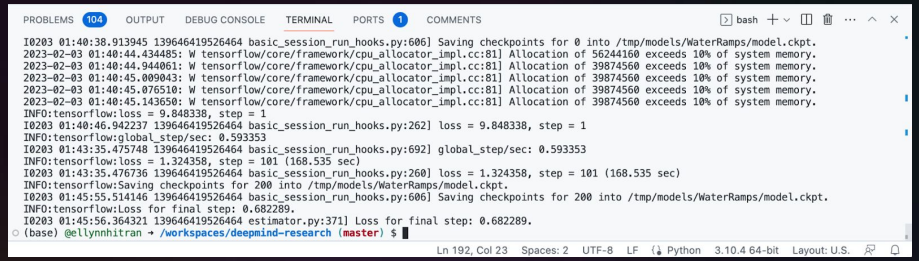
Main methodology: grid search
- Determine which hyperparameters I would fine-tune for the model: latent_size, hidden_size and decay_steps
- Train the model multiple times with selected hyperparameters.
- Evaluate the performance of the model

NOVELTY #2: HYPERPARAMETER TUNING

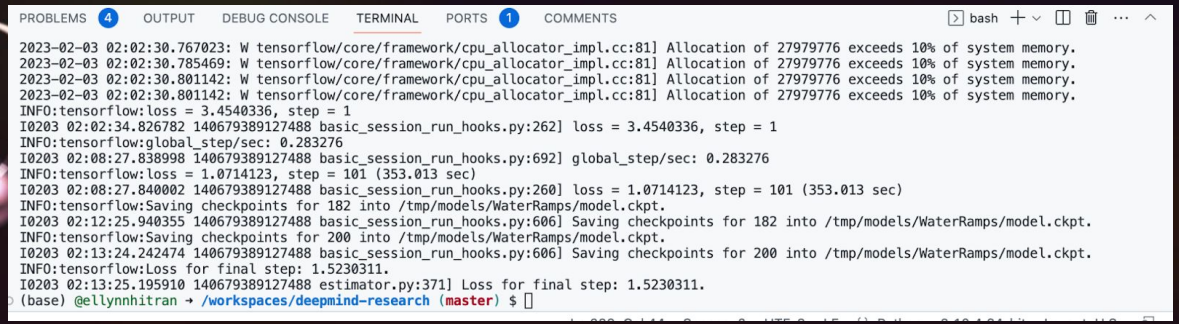
3. RESULTS

- Settings: number of steps= 2e2, batch_size = 2, number of rollouts = 5
- Base case: latent_size=128, hidden_size=128, hidden_layers=2, message_passing_steps=10

=> MSE = 0.682



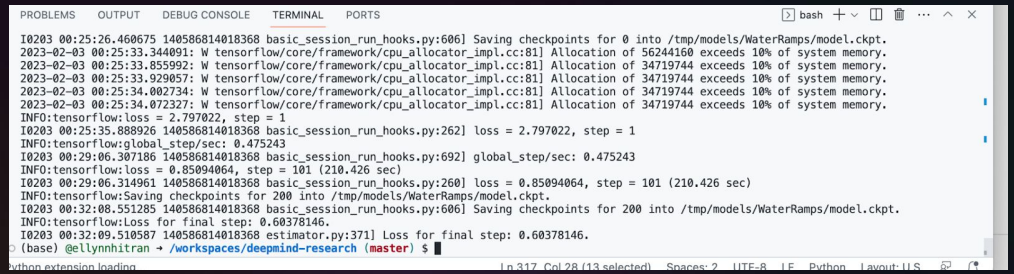
- Tune the latent size: Increase the latent size from 128 to 256
  - => MSE = 1.52



NOVELTY #2: HYPERPARAMETER TUNING

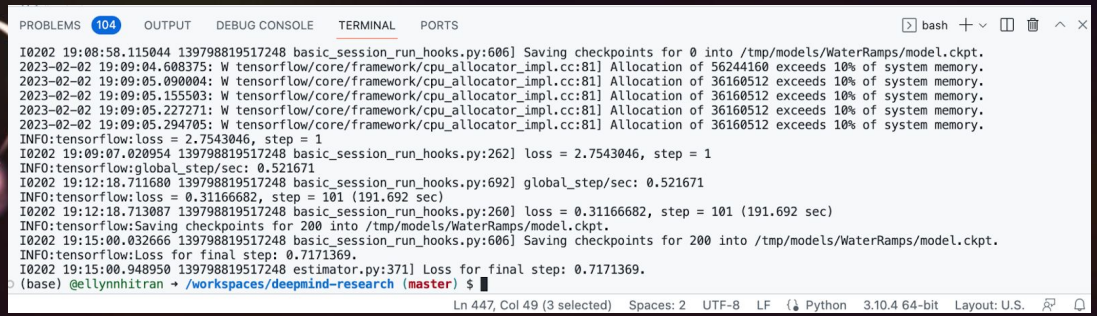
3. RESULTS

- Tune the hidden size: Increase the latent size from 128 to 256
  - => MSE = 0.60 (slightly better than base case)



- Tune the decay steps: Decrease from 5e6 to 2e2 (According to Banttgalia in 2018, we should set the decay steps proportional to the number of training examples)

=> MSE = 0.71



04

SUMMARY

Summary

COMPARISON ANALYSIS

1. Experiment with optimizers

Experiment	Mean Squared Error (MSE)	Comparison with base case
Adadelta Optimizer	3.924	+3.7 times (worse)
SGD Optimizer	3.945	+3.72 times (worse)
RMSProp Optimizer	0.885	-0.17 times (better)

2. Experiment with hyperparameters

Experiment	Mean Squared Error (MSE)	Comparison with base case
Latent size	1.52	+0.9 times (worse)
Hidden size	0.60	-0.13 times (better)
Decay step	0.71	+0.04 times (slightly worse)

CONCLUSION

1. Achievements

After 1 time changing proposal, doing the project solo, and dozens of times solving bugs

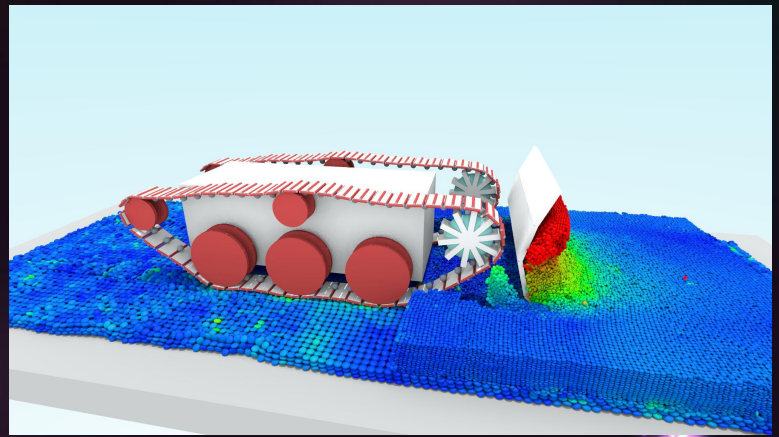
- Completed all the project requirements
- Implemented 10+ experiments and evaluated 6 of them
- Improved the training model by 13%

2. Limitations

- Scalability: Simulating physics with graph networks can be computationally expensive
- Data availability: The performance of graph networks depends on the quality and quantity of available training data.
- Model accuracy: The accuracy of graph networks in simulating complex physics may not be as high as traditional physics simulations, especially for highly complex systems.

FUTURE APPLICATIONS

- Predictive modeling and simulation of physical systems, such as fluid dynamics, solid mechanics, and heat transfer.
- Design optimization, including shape optimization and topology optimization, for example in the design of new materials or engineering components.
- Control of physical systems, for example, in robotics, autonomous vehicles, and process control.
- Prediction of material properties, such as strength, toughness, and electrical conductivity.
- Optimization of materials synthesis and processing, for example, in the development of new energy materials or biodegradable plastics.



REFERENCES

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020, September 14). *Learning to simulate complex physics with Graph Networks*. arXiv.org. Retrieved February 3, 2023, from https://arxiv.org/abs/2002.09405

Battaglia et al. (2018), "Relational inductive biases, deep learning, and graph networks", arXiv preprint arXiv:1806.01261

A comprehensive survey on Graph Neural Networks - IEEE Xplore. (n.d.). Retrieved February 3, 2023, from https://ieeexplore.ieee.org/document/9046288

"Gauge Equivariant Graph Networks for Physics Simulations" by GCP Parisi et al. published in 2020

"Graph Convolutional Networks for Complex Physical Systems" (Sanchez-Gonzalez et al., 2019)

4th inter-experiment Machine Learning Workshop. Indico. (n.d.). Retrieved February 3, 2023, from https://indico.cern.ch/event/852553/contributions/4062226/

Stanford Computer Science. (n.d.). Retrieved February 3, 2023, from https://cs.stanford.edu/people/jure/pubs/learning_to_simulate-icml20.pdf

THANK YOU FOR LISTENING

EVALUATION OF EXPERIMENTS

Experiment	Why?	Reference
Latent size	Increasing the latent size in graph networks used for simulating physics can help capture more complex relationships and patterns in the system being modeled. This can lead to improved accuracy and stability in the simulation results.	One reference for this is the paper "Gauge Equivariant Graph Networks for Physics Simulations" by GCP Parisi et al. published in 2020. The authors use graph networks with increasing latent sizes to simulate the behavior of gauge theories, which are physical systems with inherent symmetries. They show that larger latent sizes improve the accuracy and stability of the simulation results.
Hidden size	By increasing the hidden size, the network is able to capture and store more information about the physical interactions between different components in the system, allowing for more accurate and stable predictions.	One reference for this is the use of graph networks in physics simulations is the paper "Graph Networks as a Universal Machine Learning Framework for Physics" by S. Battaglia et al., published in 2018. In this paper, the authors use graph networks to simulate various physical systems, such as quantum mechanics and classical mechanics, and demonstrate that increasing the size of the hidden units in the network can improve the ability to capture complex relationships and patterns in the system.

EVALUATION OF EXPERIMENTS

Experiment	Why?	Reference
Decay step	Setting the decay steps proportional to the number of training examples can help ensure that the model training is consistent and stable, even as the number of training examples increases. By scaling the decay steps with the number of training examples, the model can gradually reduce its learning rate over time, allowing it to converge to an optimal solution.	One reference for this is the paper "Adaptive learning rate scheduling for stochastic optimization" by J. L. Ba et al., published in 2013. The authors propose an adaptive learning rate schedule that adjusts the learning rate during training based on the number of iterations or the number of training examples processed. They show that this approach can lead to improved convergence and performance compared to fixed learning rate schedules.
SGD Optimizer	Stochastic gradient descent (SGD) is a popular optimization algorithm for training machine learning models, including graph networks used for simulating physics. One reason for using SGD is that it can efficiently handle large-scale datasets, which is often the case in physics simulations.	A reference for using SGD in simulating physics with graph networks is the paper "Gauge Equivariant Graph Networks for Physics Simulations" by GCP Parisi et al., published in 2020. The authors use SGD to train graph networks to simulate the behavior of gauge theories, which are physical systems with inherent symmetries. They show that SGD is effective in optimizing the network parameters and leads to accurate and stable simulation results.

EVALUATION OF EXPERIMENTS

Experiment	Why?	Reference
Adadelta Optimizer	Adadelta is a popular optimization algorithm that is well suited for training deep neural networks. It is a stochastic gradient descent optimization algorithm that adapts the learning rate based on the historical gradient information.	According to Zeiler in 2012, Adadelta is particularly useful in situations where the gradients are noisy or when the scale of the gradients changes during training. This can be especially beneficial in physics simulations where the gradients are often uncertain or unpredictable.
RMSProp Optimizer	RMSProp is an optimization algorithm that is used for training deep neural networks. Like Adadelta, it is a stochastic gradient descent optimization algorithm that adjusts the learning rate based on the historical gradient information.	According to Tieleman, T., & Hinton, G. (2012), RMSProp is well suited for simulating physics with graph networks because it can handle the non-stationarity of the gradients, which is often the case in physics simulations. It does this by dividing the historical gradient information by an exponentially decaying average of the squared gradients. This allows RMSProp to dynamically adjust the learning rate on a per-parameter basis, which can help stabilize the training process and improve the accuracy of the results.