UNIVERSITY OF WATERLOO
AI TRAINING

Lecture Module

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

CONTENTS

Chapter 1. Understanding Deep Learning, Text Generation, And Transformer Architectures

Maryam Dialameh, PhD Student

Hossein Rajabzadeh, PhD Candidate

Mechanical and Mechatronics Engineering

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

OUTLINE

1. Text Generation
2. Deep Learning Models for Sequential Data
3. Transformers: Decoder Only models

Introduction

- **Examples of Large Language Models (LLMs) and Vision Language Models (VLMs)**

Video Generation

Text Generation

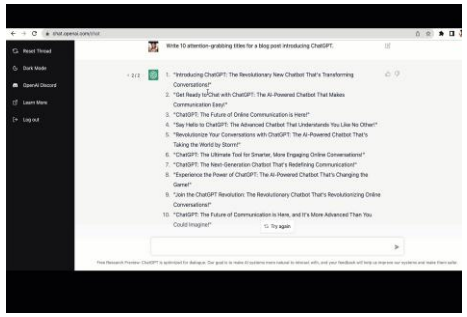
Image Generation

Robotics and Planning

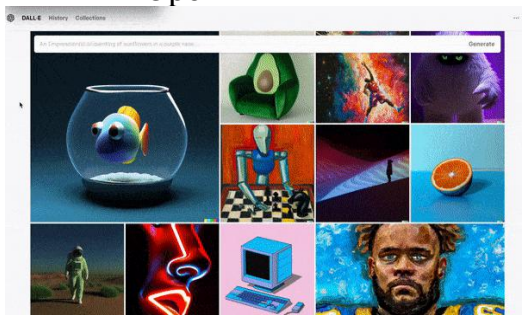
...

OpenAI - Sora

OpenAI - ChatGPT



OpenAI-DALL-E



Google-DeepMind: RT-2

Introduction

LLMs For Robotics

Ref: https://arxiv.org/pdf/2401.04334

Instruction: pick green chip bag from bottom drawer and place on counter

Environment + Instruction

Predefined Action Pool

```
move_hand()
grasp_object()
release_object()
...
```

GPT-4V

Reasoning

Planning

GPT-4V Generated Task Plan

Step by Step Instruction

1. move hand to the location of the target in the bottom drawer
2. grasp the target
3. move hand with the target to the counter
4. release the target onto the counter

Action Plan

```
move_hand(bottom drawer)
grasp_object(target)
move_hand(counter)
release_object(target)
```

Target Object

green jalapeno chip bag

Environment State

<green jalapeno chip bag> in the <bottom drawer>

↓

<green jalapeno chip bag> on the <counter>

Environment + Instruction

pick anything

Prediction by GPT-4V

Step by Step Instruction

1. Move the robot hand above an object to pick it
2. Grasp the chosen object
3. Move the hand to a different location with the object grasped

Action Plan

move_hand(above an object)
grasp_object(any object)
move_hand(different location)

Target Object: any object

Environment State

A work area containing <various objects>

↓

<an object> moves it to a <different location>

Matching Score: 8/10

Environment + Instruction

open the drawer

Prediction by GPT-4V

Step by Step Instruction

1. Move the robot hand to the handle of the drawer and pull it open

Action Plan

move_hand(the handle of the drawer)
move_hand(right end)

Target Object: drawer

Environment State

<drawer> closed

↓

<drawer> open

Matching Score: 10/10

What is sequential data?

- Sequential data refers to data points that are collected or recorded in a specific order over time.

- Here are some main categories of sequential data:

1.Time Series Data

1. Examples: Stock prices, weather data, sensor readings, economic indicators.

2.Text Data

1. Examples: Sentences, paragraphs, documents, and transcripts.

3.Audio Data

1. Examples: Speech recordings, music tracks, sound waves.

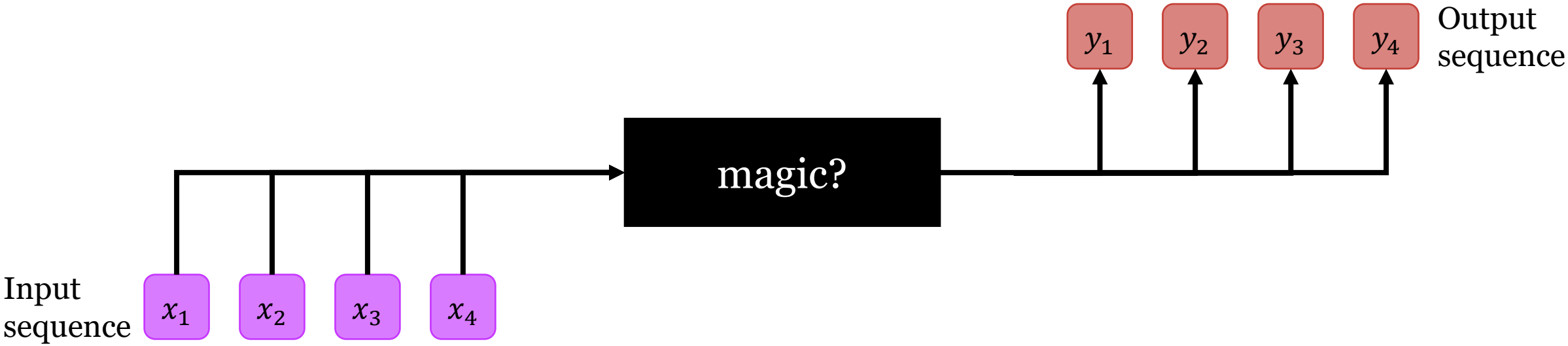
4.Video Data

1. Examples: Sequences of frames in a video, video streams.

1. Text Generation

Text Generation: Sequence to Sequence

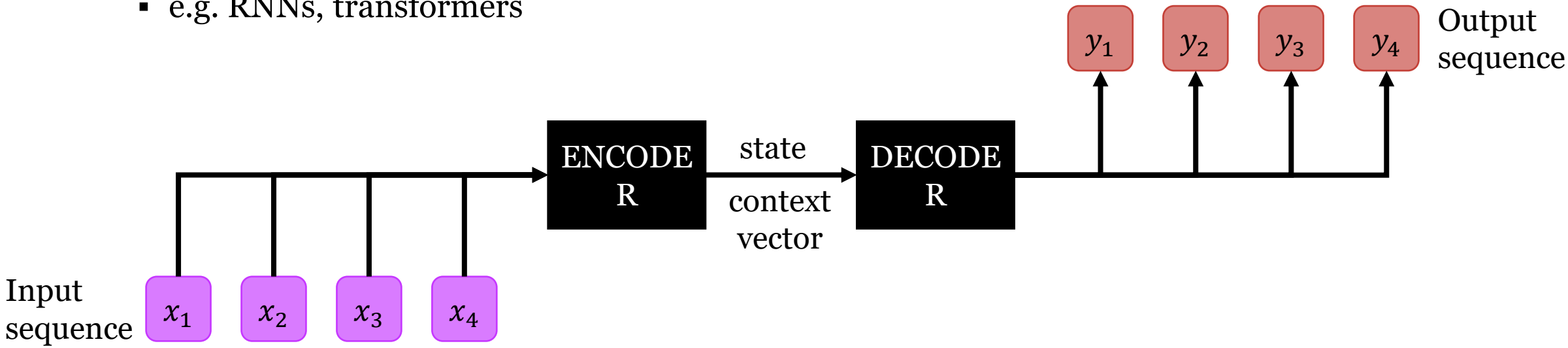
- **Example Scenarios**
  - Text → Text (e.g. Q/A, translation, text summarization)
  - Image → Text (e.g. image captioning)



Sequence to Sequence

- **Example Scenarios**
  - Text → Text (e.g. Q/A, translation, text summarization)
  - Image → Text (e.g. image captioning)

- **How?** Usually Encoder-Decoder models
  - e.g. RNNs, transformers



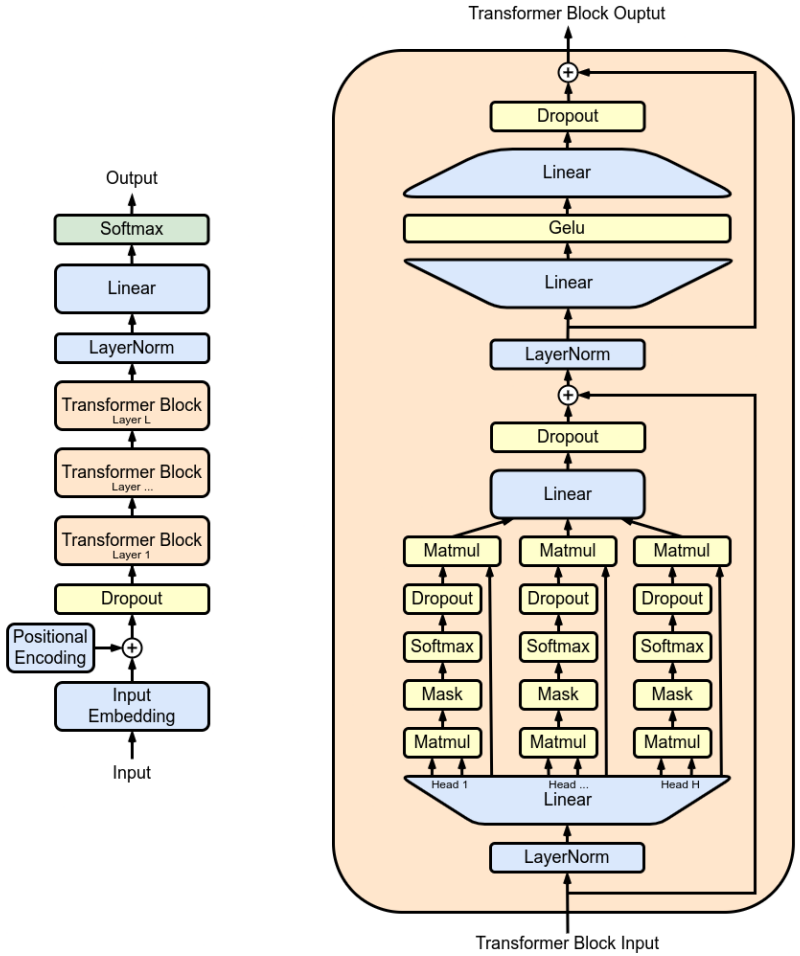
2. Deep Learning Models for Sequential Data

What is Large Language Model (LLM)?[1]

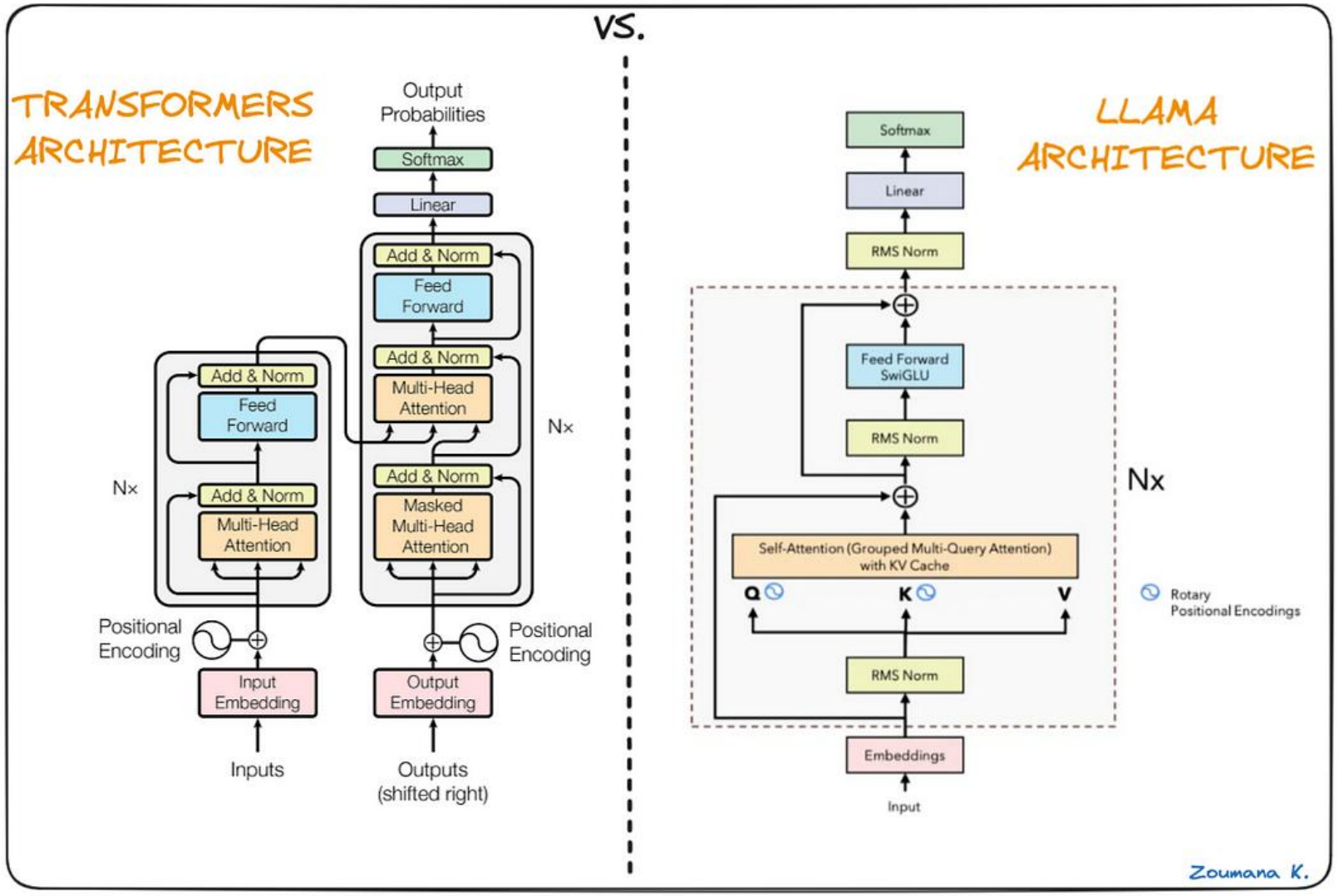
- Language models are computational models that have the capability to understand and generate human language.
- Deep learning algorithm that can perform various NLP tasks
- Are trained on massive datasets that allow them to recognise, translate, predict, or generate text or other content
- Unsupervised multi-task learners

1. Chang, Y., Wang, X., Wang, J., Wu, Y., Zhu, K., Chen, H., Yang, L., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., & Xie, X. (2023). **A Survey on Evaluation of Large Language Models.** *ArXiv, abs/2307.03109*.

GPT-3 Architecture

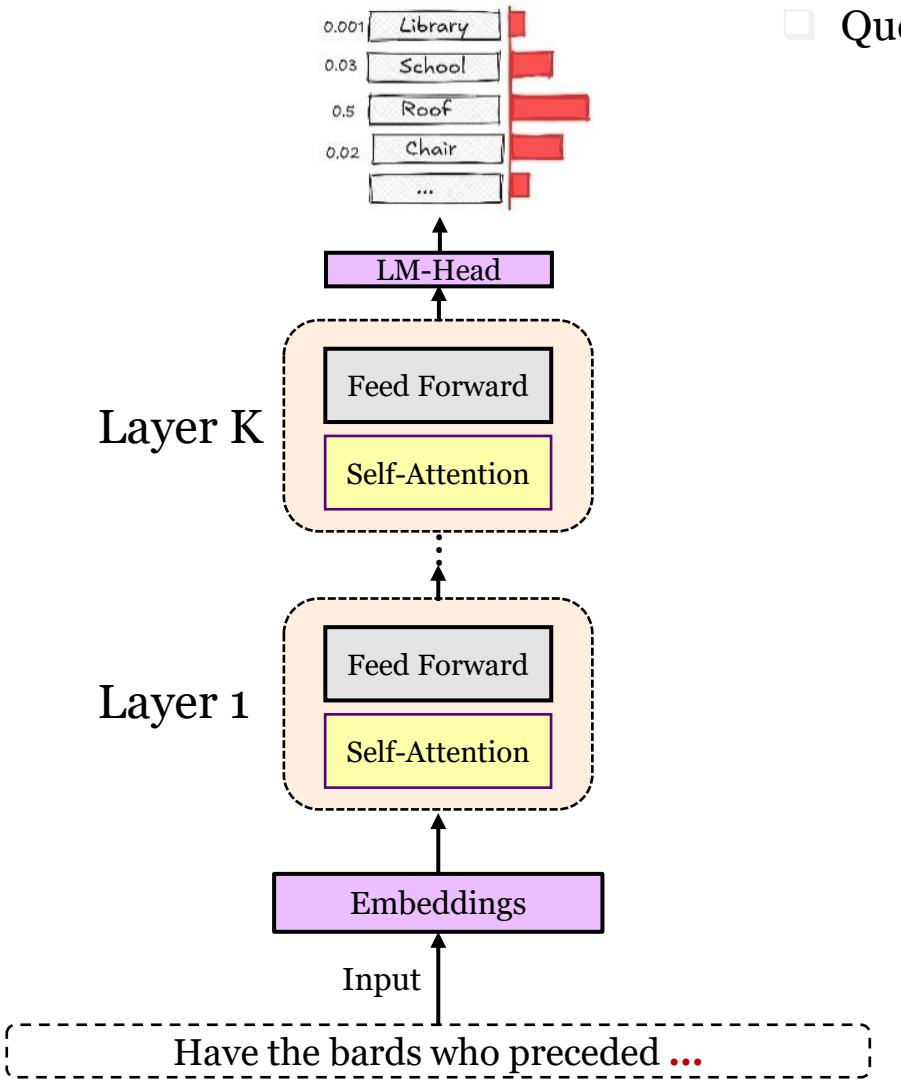


LLAMA Architecture



3. Transformers: Decoder Only models

Decoder-Only Models



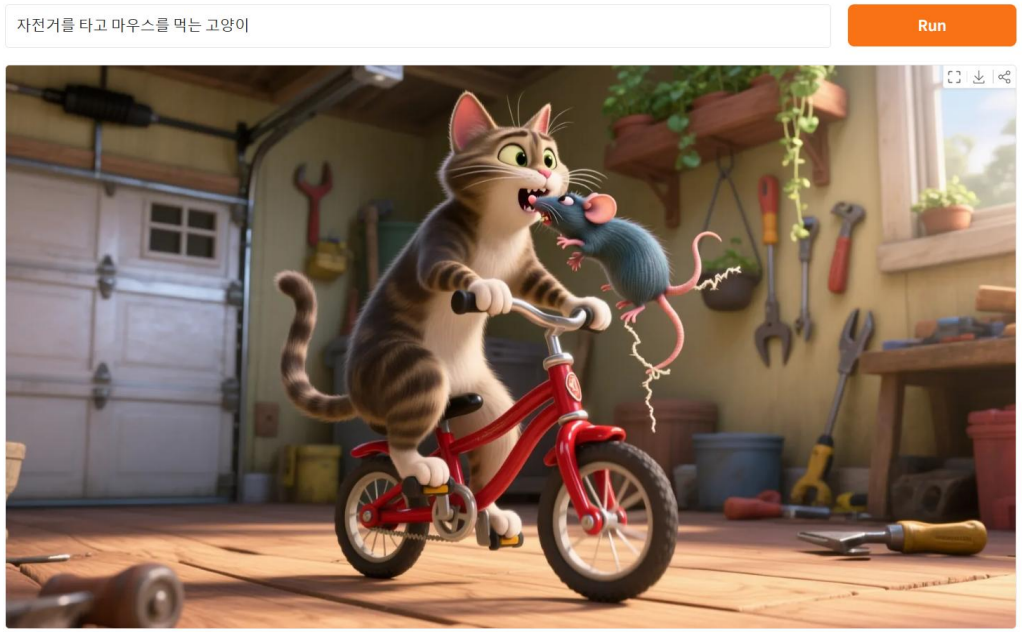
- Question: How does an LLM generate next token given previous tokens?

Workshop

- Google Colab

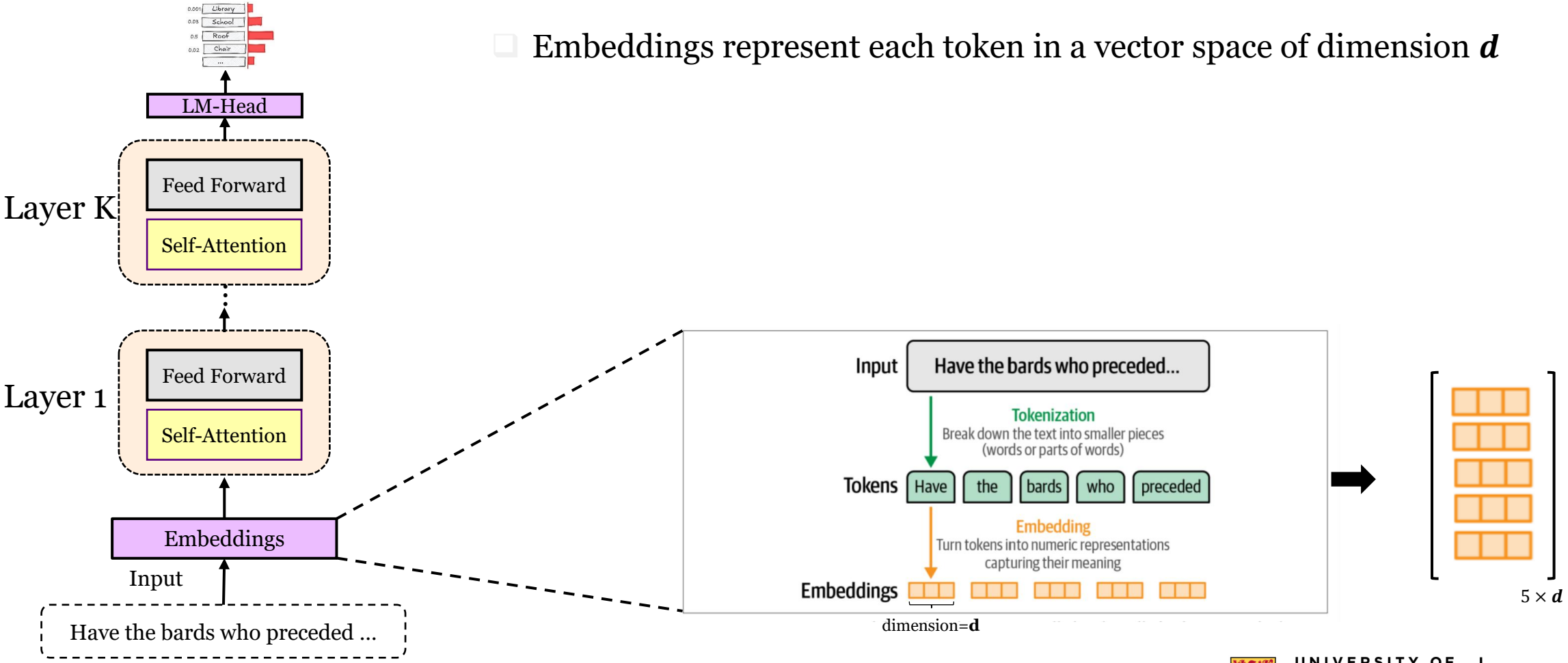
```
from diffusers import StableDiffusionPipeline, EulerDiscreteScheduler
import torch

# model_id = "stabilityai/stable-diffusion-xl-base-1.0"
model_id = "stabilityai/stable-diffusion-2"  # smaller model
scheduler = EulerDiscreteScheduler.from_pretrained(model_id,
subfolder="scheduler")
pipe = StableDiffusionPipeline.from_pretrained(model_id, scheduler=scheduler,
torch_dtype=torch.float16)
pipe = pipe.to("cuda")

prompt = "a photo of cat riding a bike."
image = pipe(prompt).images[0]

# Save the generated image
output_path = "image.png"
image.save(output_path)
print(f"Image saved at {output_path}")
```

- Playground
  - A powerful text to image model (QWEN-Image) https://huggingface.co/spaces/Qwen/Qwen-Image



Introduction

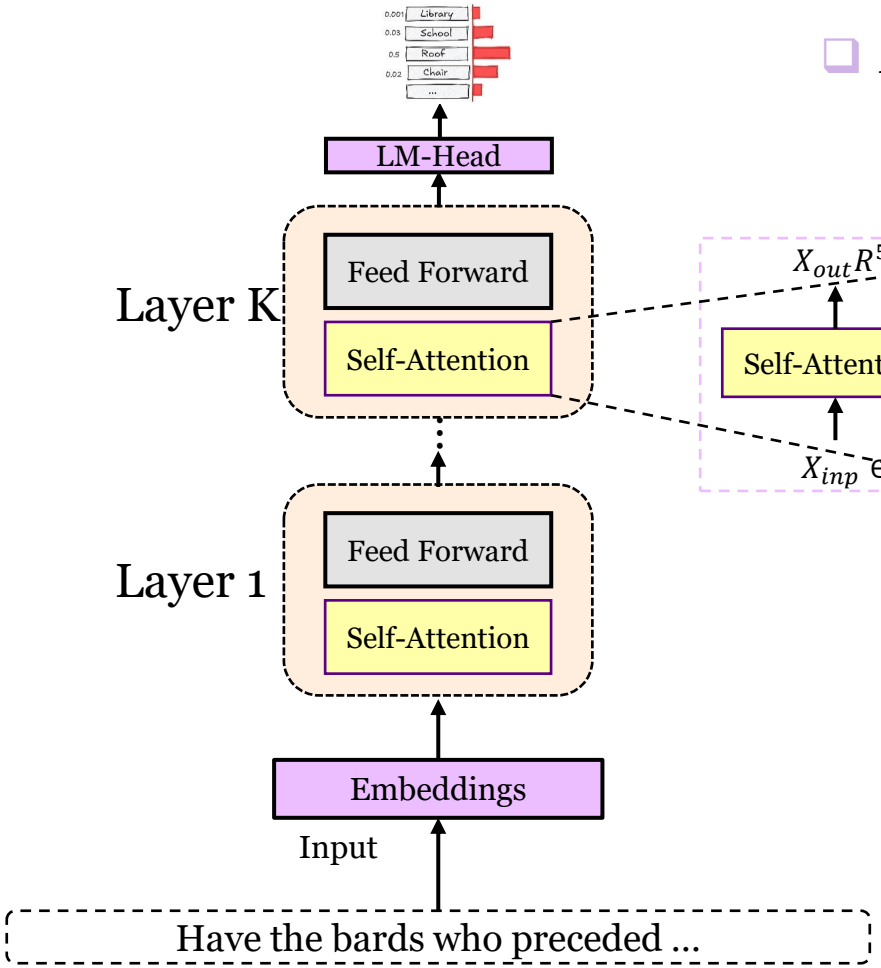
- Question: How does an LLM generate next token given a previous tokens?



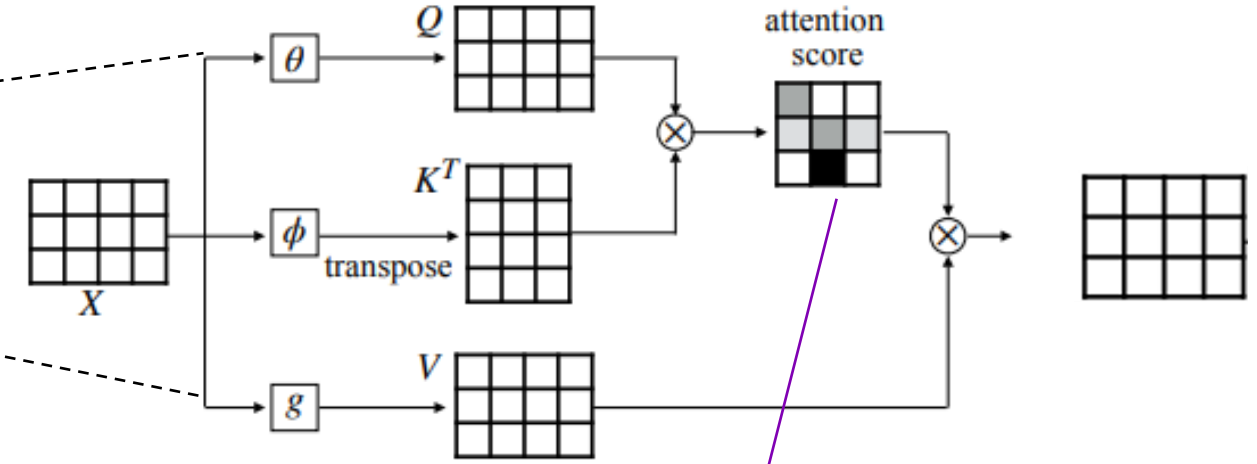
- Embeddings represent each token in a vector space of dimension d

Introduction

- Question: How does an LLM generate next token given a previous tokens?



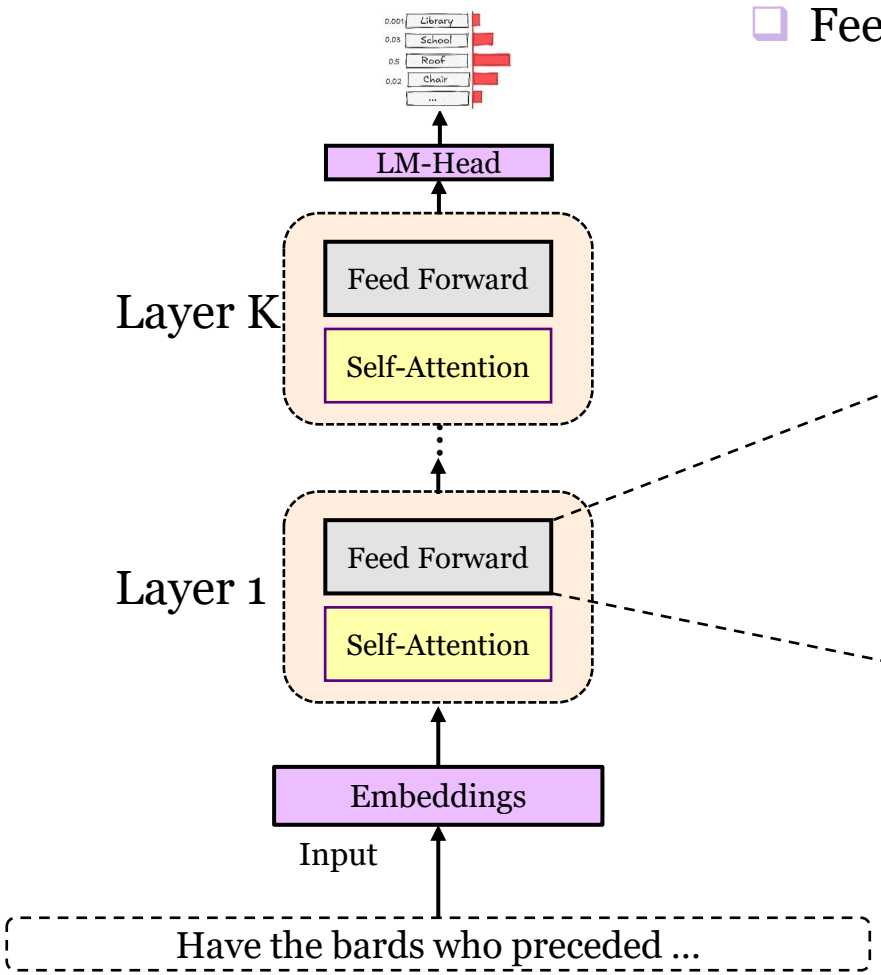
- Attention is like token mixing using weighted combination of tokens.



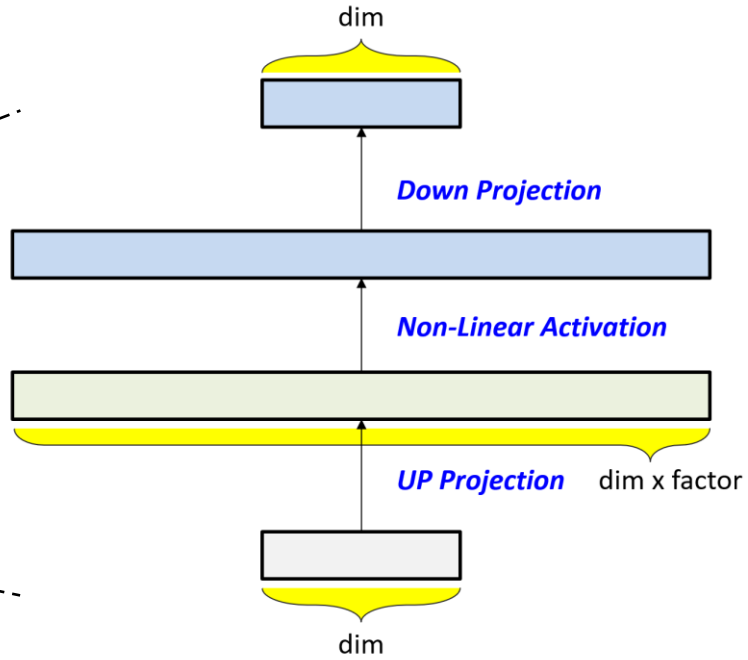
	Have	the	bards	who	preceded
Have	1				
the	0.2	0.6			
bards	0.6	0.5	0.6		
who	0.4	0.2	0.6	0.8	
preceded	0.7	0.3	0.5	0.4	0.7

Introduction

- Question: How does an LLM generate next token given a previous tokens?

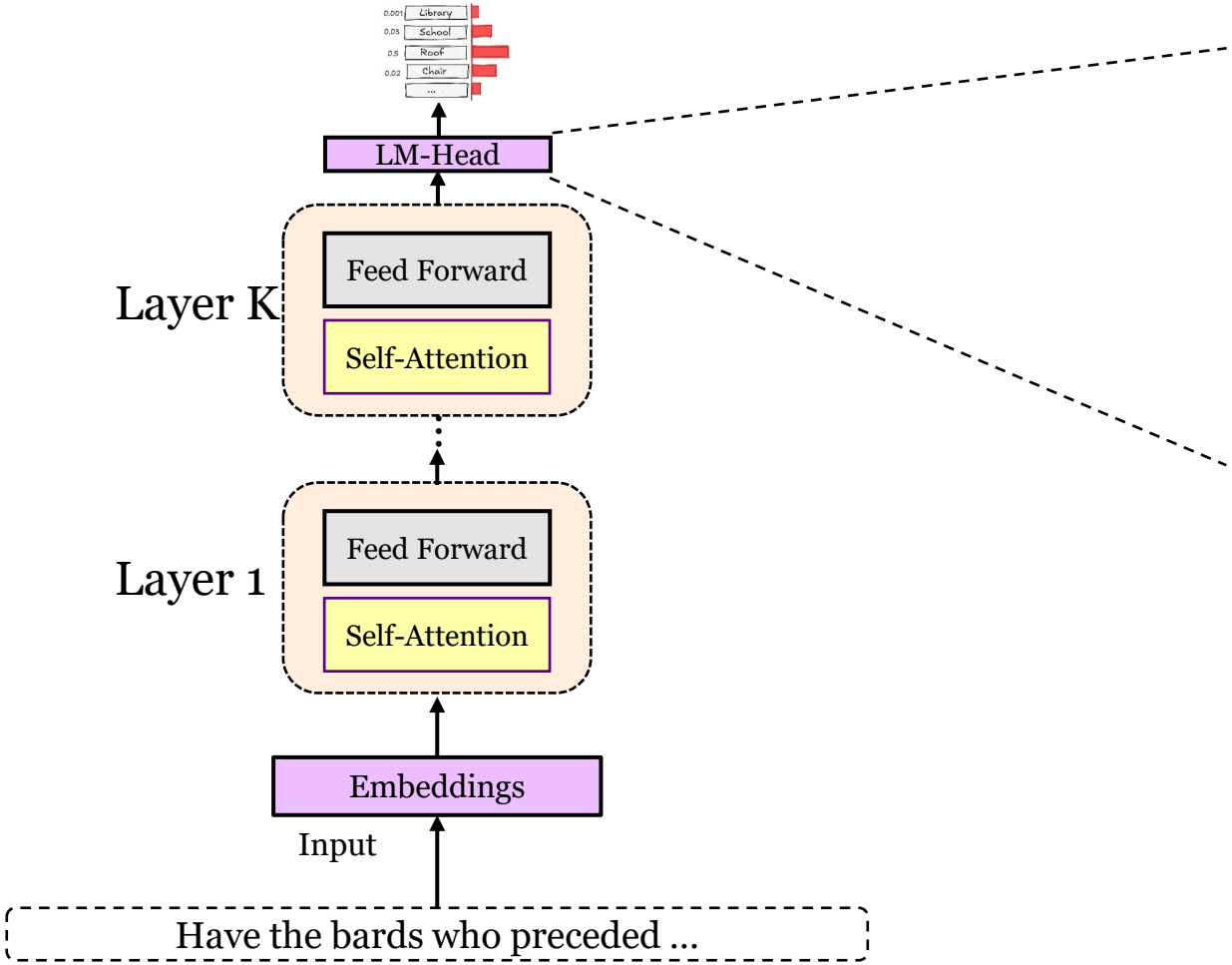


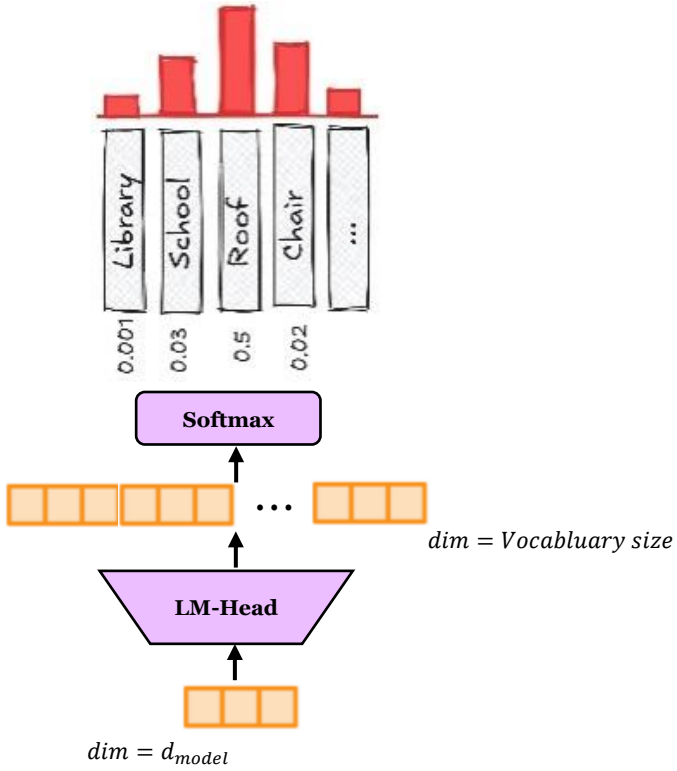
- Feed Forward is like a token transformation module.



Introduction

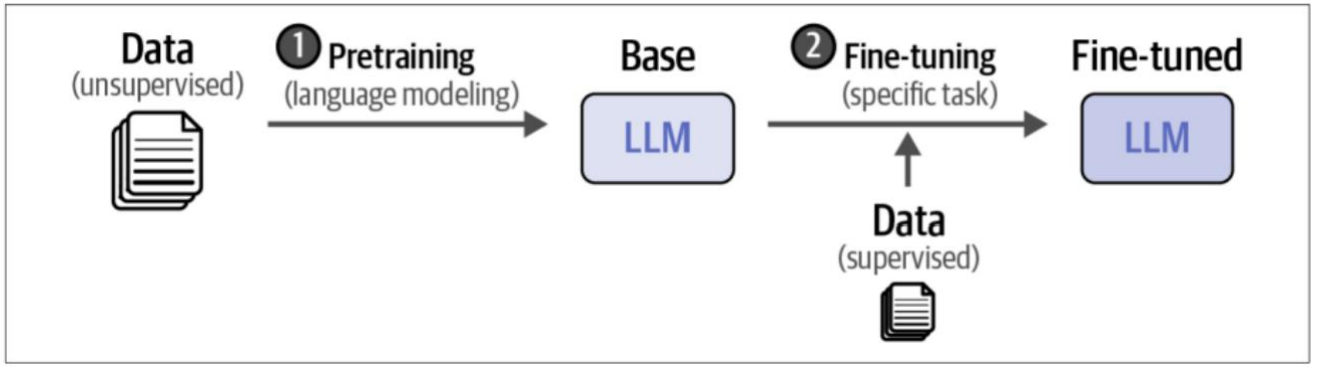
- Question: How does an LLM generate next token given a previous tokens?





Some Terminologies

- **Defining Training and Finetuning**



Pre-Training/FineTuning/Alignment in LLMs

1 Pretraining

Dataset:
100B to >5T tokens

Task: Next-token prediction on unlabeled texts

Output: base model / "foundation model"

Project Gutenberg (PG) is a volunteer effort to digitize and archive cultural works, as well as to "encourage the creation and distribution of eBooks." It was founded in 1971 by American writer Michael S. Hart and is the oldest digital **library**. Most of the items in its collection are the full texts of books or individual stories in the public domain. All files can be accessed for free under an open format layout, available on almost any computer. As of 3 October 2015, Project Gutenberg had reached 50,000 items in its collection of free eBooks.

2 Supervised finetuning

More next-token prediction

Usually 1k-50k instruction-response pairs

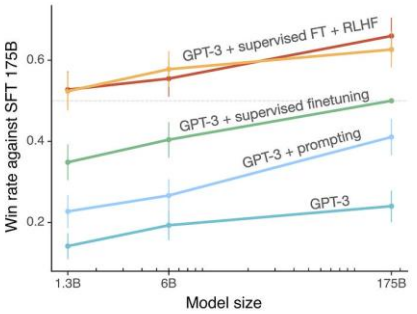
```
{
    "instruction": "Write a limerick about a
                    pelican.",
    "input": "",
    "output": "There once was a pelican so fine,
              \nHis beak was as colorful as
              sunshine,\nHe would fish all day,\nIn
              a very unique way,\nThis pelican was
              truly divine!\n\n\n"
},

{
    "instruction": "Identify the odd one out from
                    the group.",
    "input": "Carrot, Apple, Banana, Grape",
    "output": "Carrot\n\n"
},
```

3 Alignment

Align with human preferences

Usually reinforcement learning with human feedback (RLHF)

>50k examples



Chapter 2. Computer Vision Models

Maryam Dialameh, PhD Student

Hossein Rajabzadeh, PhD Candidate

Mechanical and Mechatronics Engineering

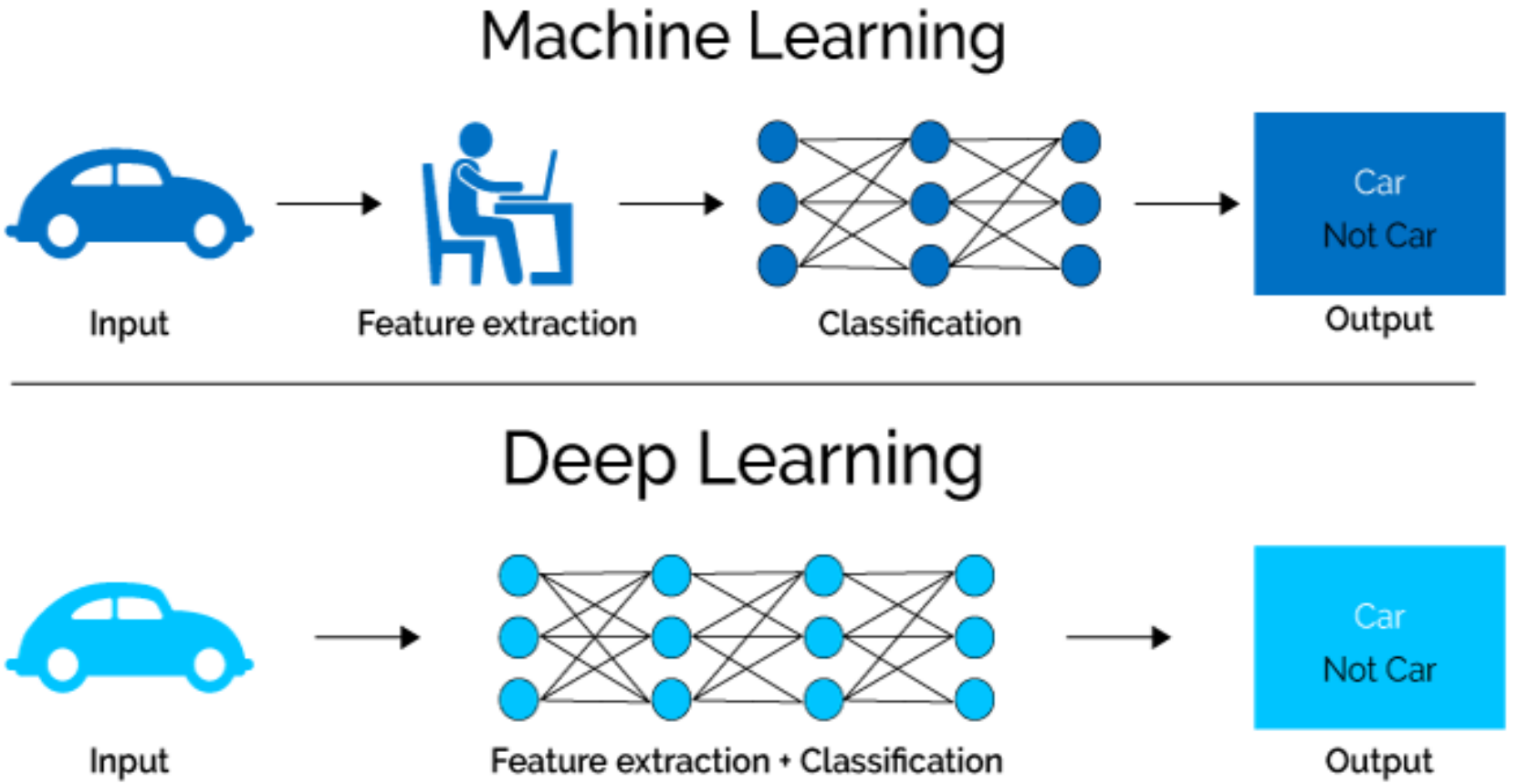
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

OUTLINE

1. Machine Learning VS Deep Learning
2. Basics of CNN
3. YOLO: You Look Only OncePINNs Exercises
4. Diffusion Models

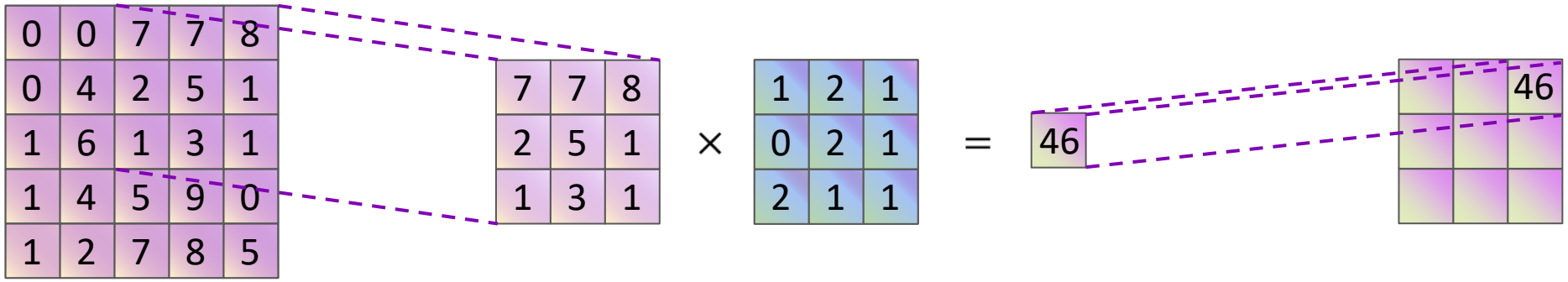
Machine Learning VS Deep Learning

- Machine Learning -> Deep Learning

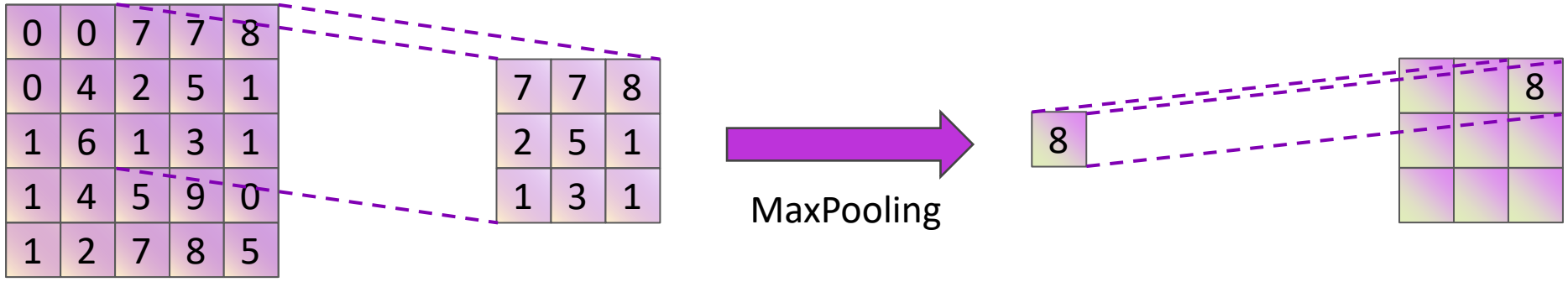


Basics of CNN

❿CNN operation

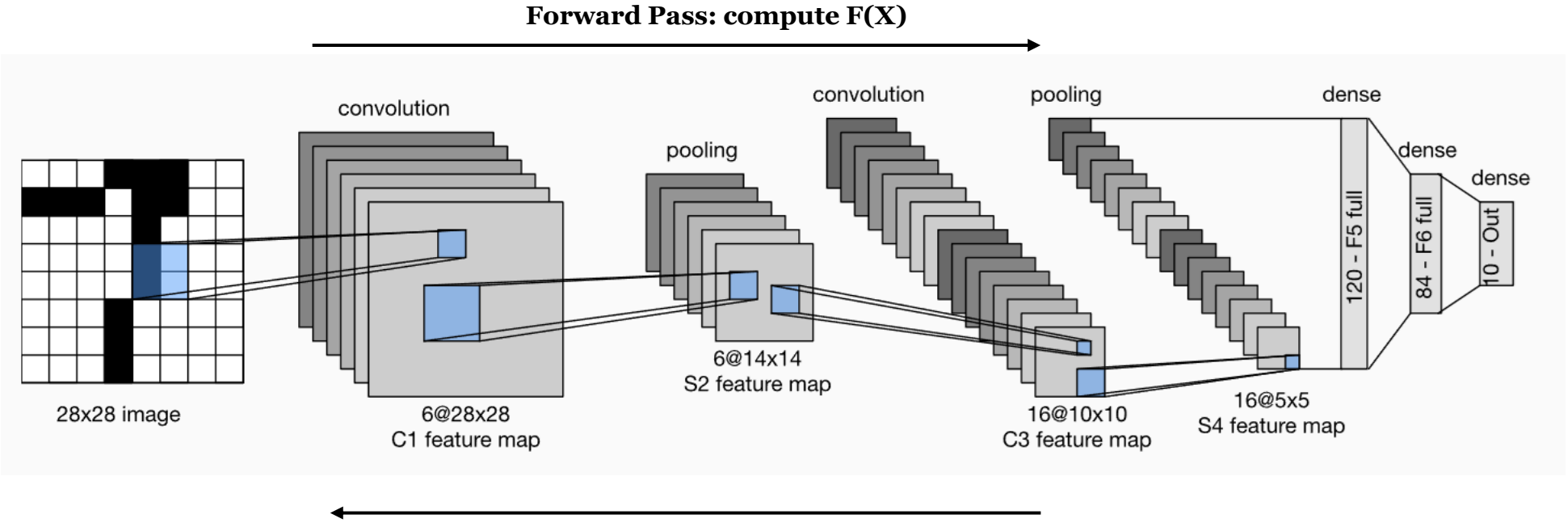


❿Pooling operation



LeNet-5

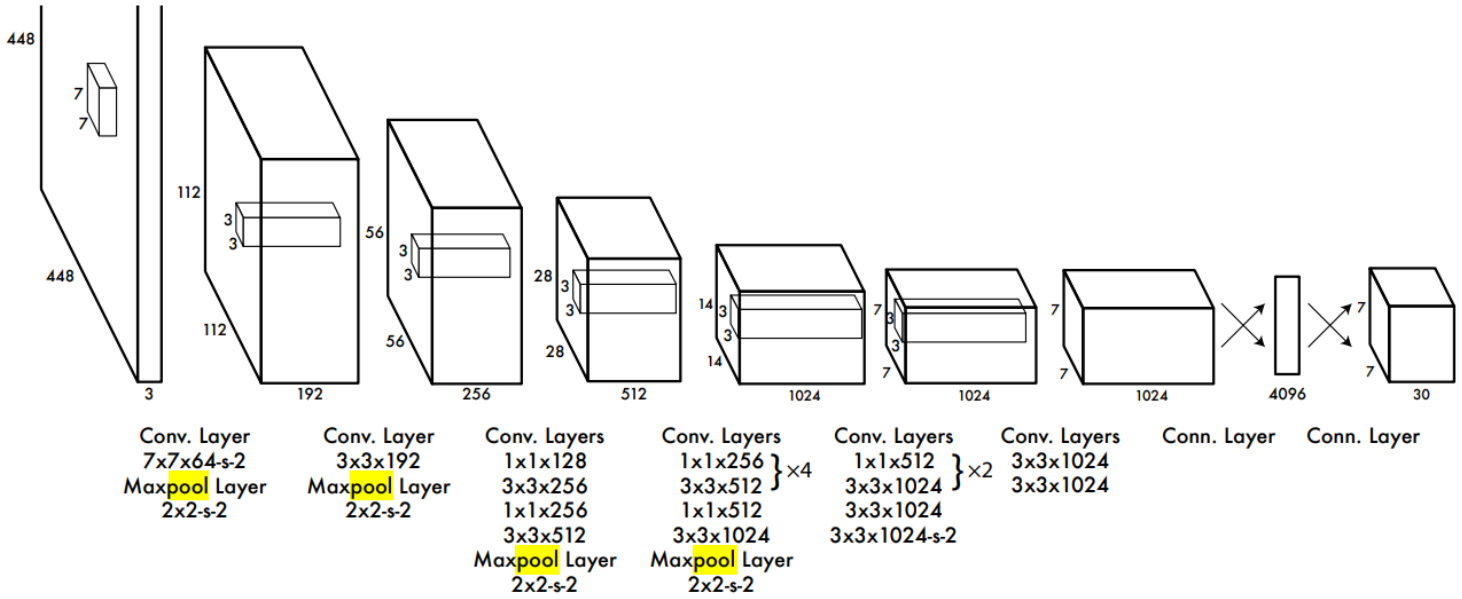
- ❿ Sigmoid activation after each convolution operation
- ❿ Using average pooling

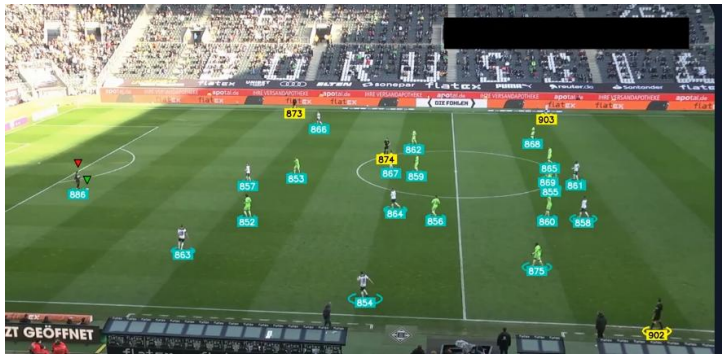


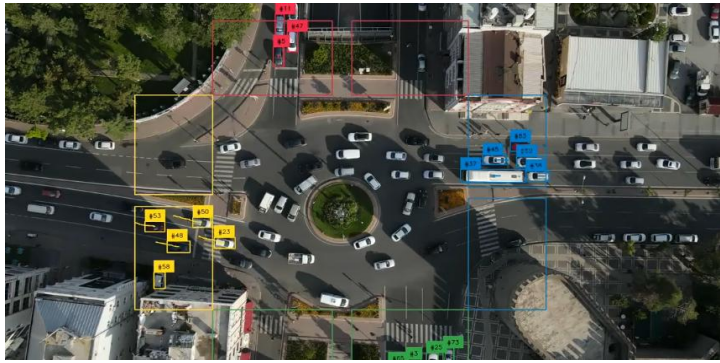
YOLO: You Look Only Once

YOLO (You Only Look Once) is a real-time object detection model that treats detection as a single regression problem, directly predicting **bounding boxes** and **class probabilities** from an entire image in **one forward pass** of the network.

- The image is divided into a grid, and each grid cell predicts a fixed number of bounding boxes and confidence scores.
- This makes YOLO **fast** (suitable for real-time use) and **end-to-end trainable**, unlike older methods that used separate stages for region proposal and classification.
- YOLO is widely used in applications like surveillance, robotics, and autonomous driving because it balances **speed and accuracy**.







UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Assignment#1: Real-Time Object Detection with YOLO

🔍 **Problem**

Detect and label objects from a live laptop camera feed in **real time**

🧠 **Model**

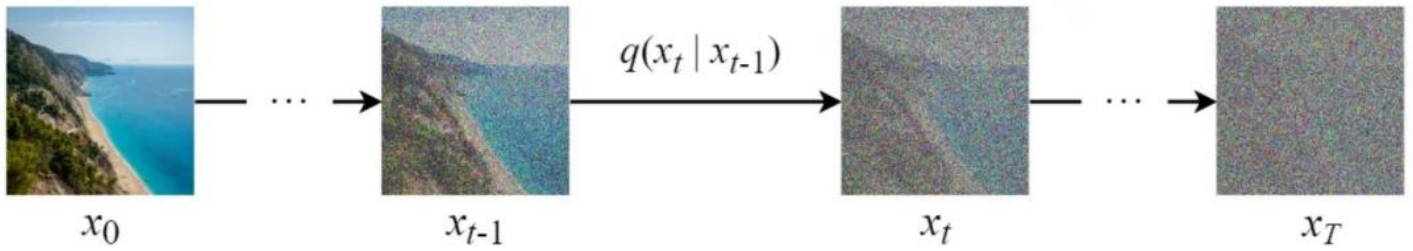
- YOLOv8n (You Only Look Once – nano version, CPU-friendly)
- Pretrained on COCO dataset (80 common object classes)
- Single forward pass → Bounding boxes + Class labels + Confidence scores

jects from a live laptop camera feed in real time.

📦 Dependencies: *pip install ultralytics opencv-python*

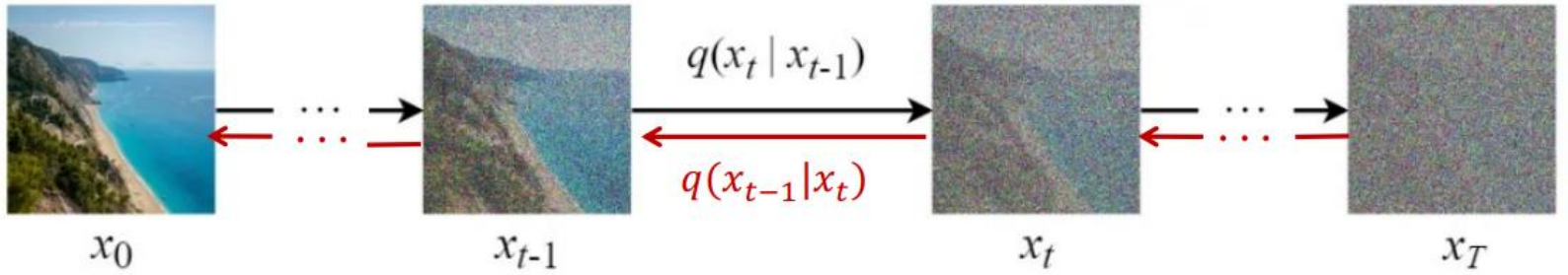
Diffusion Models

Forward Diffusion Process



Reverse Denoising Process

From Steins (medium.com)



Assignment #2: Image Classification

- Problem: We are given a dataset of 60K images ranged in 10 classes. Our task is to implement a CNN model with the following details to classify this dataset.
- Model description:
  - Model 1: Multilayer Perceptron (Feed Forward Network)
    - Number of hidden layer: your choice
    - Activation Function: ReLU
    - DropOut size: 0.2
  - Model 2: Convolutional Neural Network
    - Number of convolution layer: your choice
    - Activation Function: ReLU
    - Number of convolution filter: your choice
    - Filter size: 3 by 3
    - DropOut size: 0.2
- Optimizer: SGD
- Loss Function: CrossEntropy

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Chapter 3. Physics-Informed Neural Network

Salman Lari, PhD Candidate
Mechanical & Mechatronics Engineering

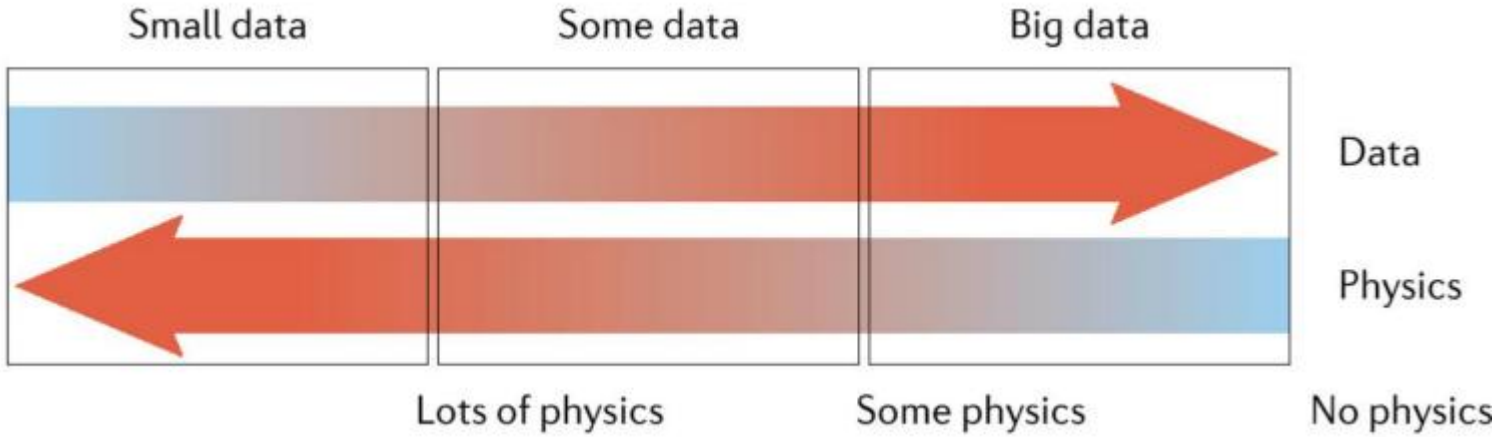
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

OUTLINE

1. Introduction
2. Traditional Physics Informed Neural Networks (PINNs)
3. Applications of PINNs Inverse Problems
4. PINNs Exercises
5. Physics Informed DeepONets
6. Physics Informed DeepONet Exercises

1. Introduction

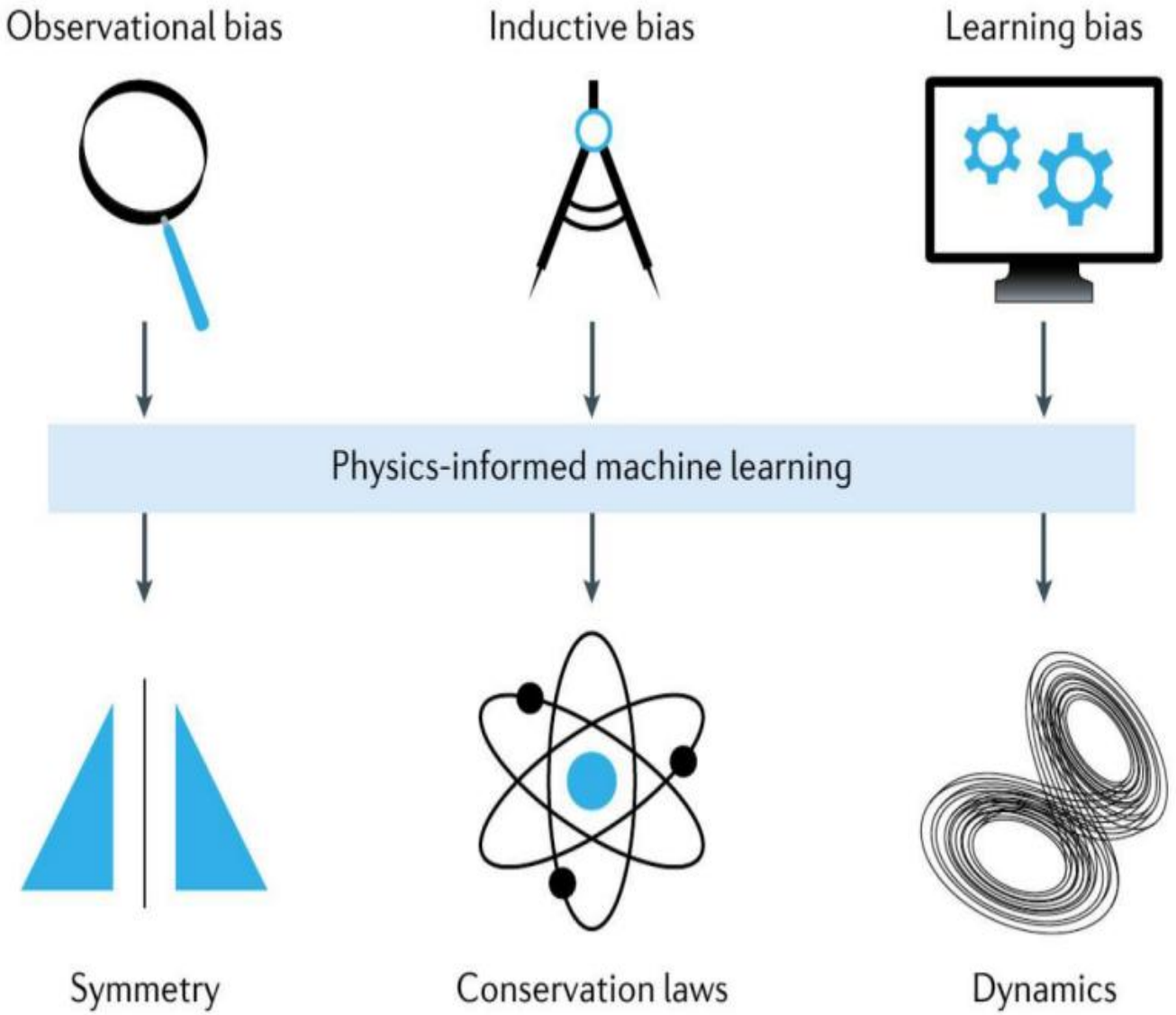
- Deep learning has expanded in recent years
  - Availability of big data
  - Improvements in hardware (GPU/TPU)
  - Open source libraries
- Data is not always available in all cases
  - Expensive
  - Time consuming
- But may have good physical understanding of system
  - Scientific experiments
  - Hardware design
- **Solution: Include physics of problem into neural networks to train with much less data**



Korniadakis et al 2021

How Can Physics Help?

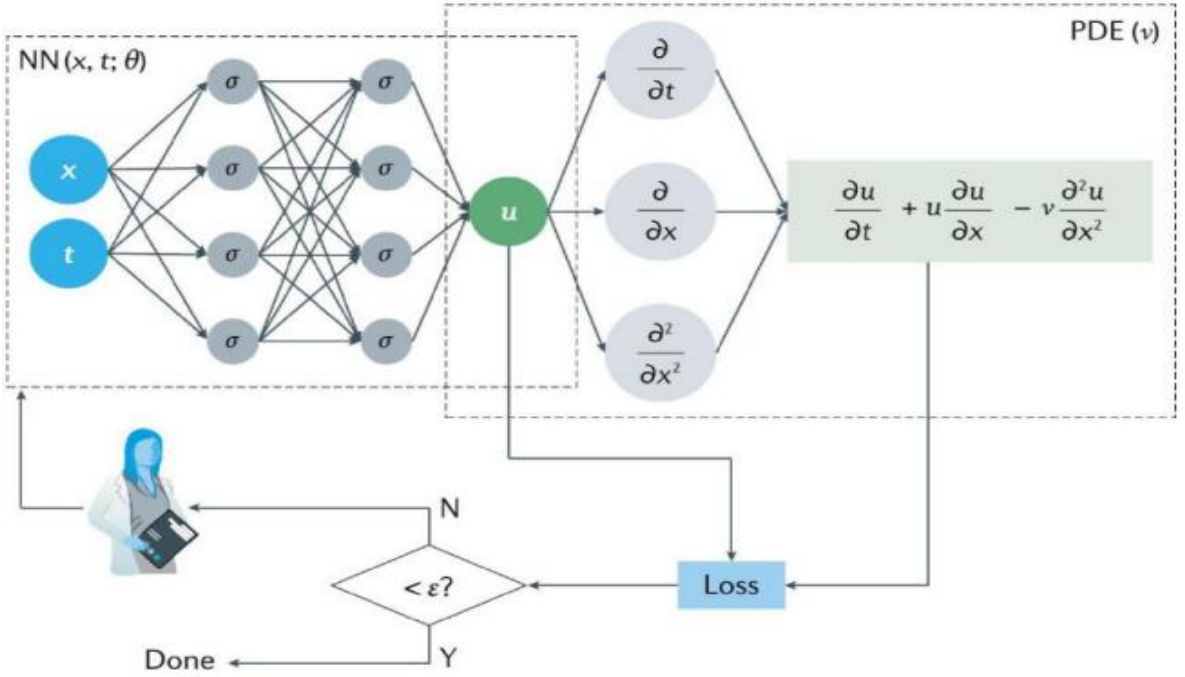
- Neural networks retain bias of its training data
  - Gender bias in NLP (Sun et al 2019)
  - Racial and age bias in image recognition (Nagpal et al 2019)
- Some bias can be removed
  - Data augmentation
  - Increase data quantity
- Physics knowledge can remove data biases system with well understood physics
  - Symmetries
  - Conservation laws
  - Partial differential equations (PDEs)



Korniadakis et al 2021

How to Encode Physics into Neural Networks

- Add known physical laws into loss function
  - Introduces soft constraints
  - Improves with more training
- Encode derivatives by employing automatic differentiation
  - Accurate
  - Fast
- Weight data and physical laws to improve training
- May need second derivatives → No ReLU activation function
- Normalize equations



$$\mathcal{L} = w_{data}\mathcal{L}_{data} + w_{PDE}\mathcal{L}_{PDE},$$

where

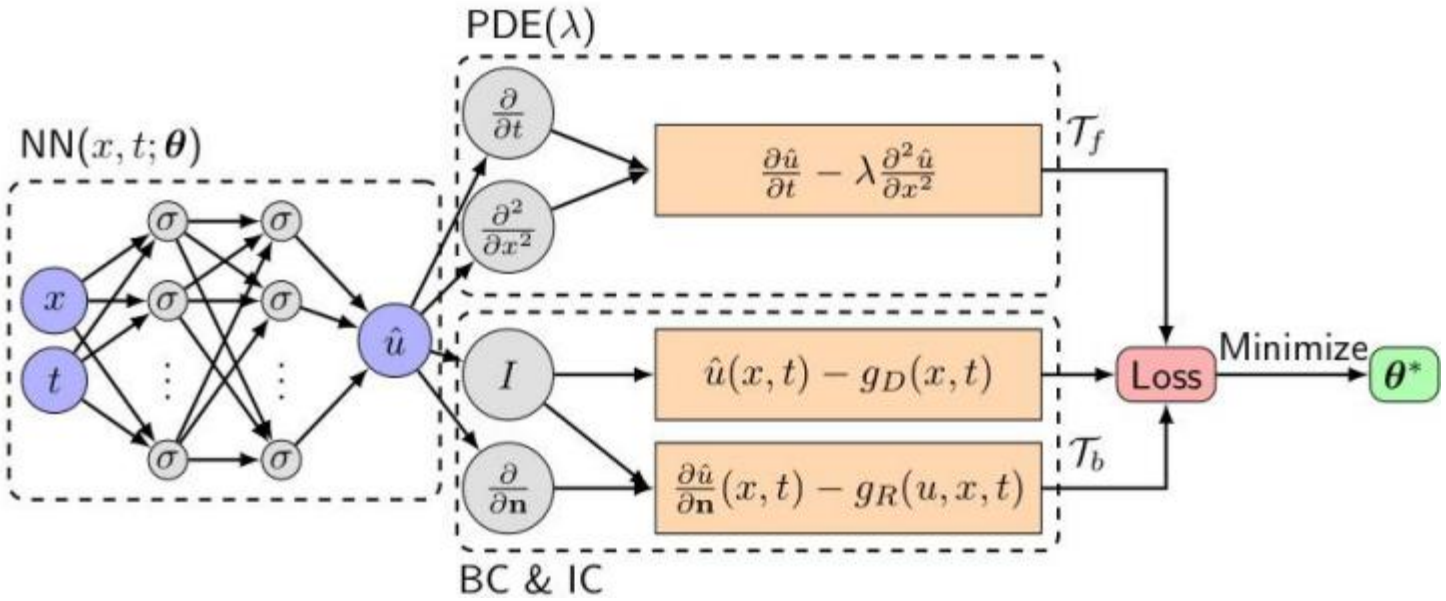
$$\mathcal{L}_{data} = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (u(x_i, t_i) - u_i)^2 \quad \text{and}$$

$$\mathcal{L}_{PDE} = \frac{1}{N_{PDE}} \sum_{j=1}^{N_{PDE}} \left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} \right)^2 |_{(x_j, t_j)}$$

Korniadakis et al 2021

2. Traditional Physics Informed Neural Networks (PINNs)

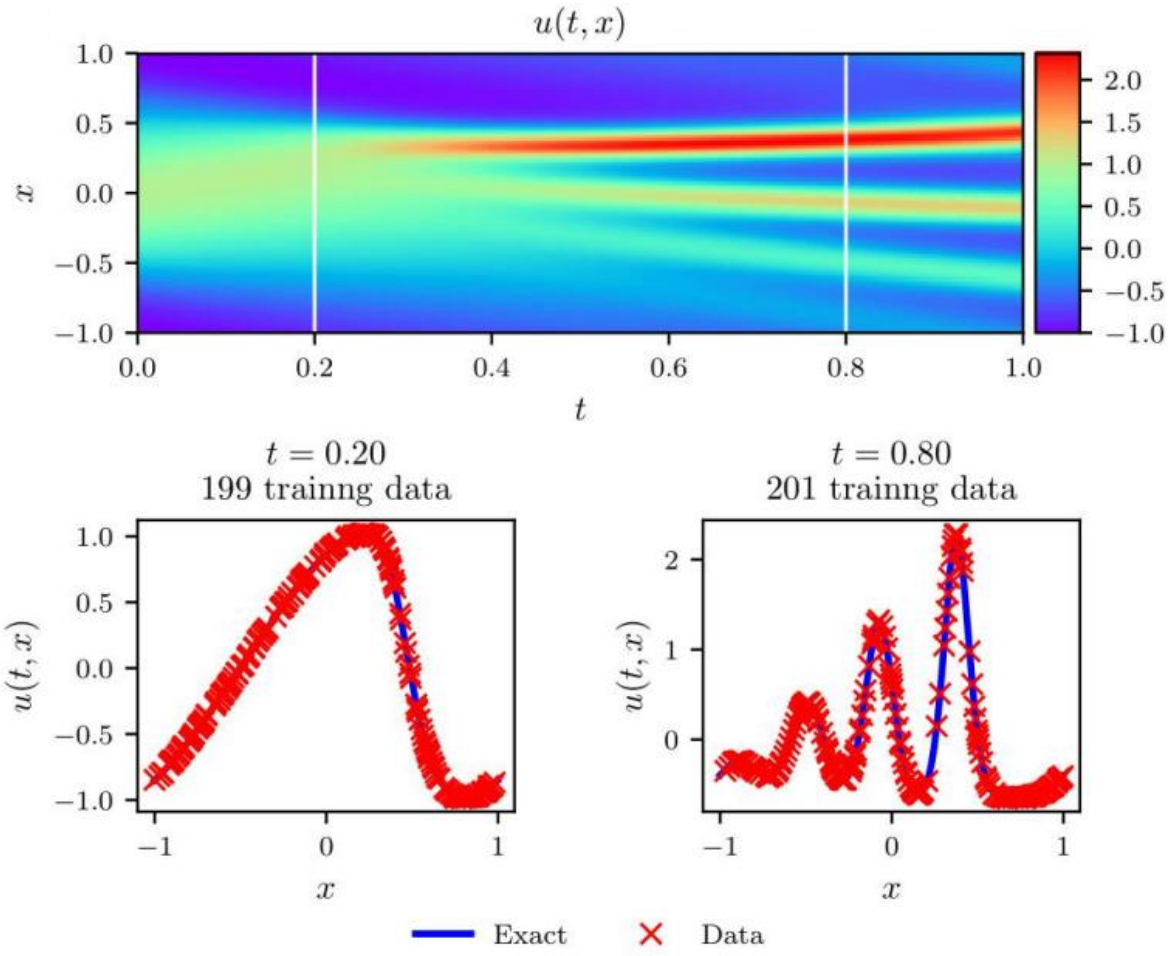
- PINNs are the most well known type of physics informed deep learning models
- Inputs
  - Coordinates (space and/or time)
  - May add auxiliary variables to input
- Outputs
  - PDE solution fields
  - May add other outputs (inverse problems)
- Train by constraining encoded physics
  - Randomly sample domain
  - May add known data
- **Trained for a single case**
  - 1 set of ICs/BCs
  - 1 set of PDEs → cannot modify source terms



Lu et al 2019

Types of Problems for PINNs

- Forward problems
  - Solve PDEs within specified domain
  - We will look at using PINNs to solve various forward problems
- Inverse problems
  - Given data that obeys a known (or partially known) PDE
  - Compute quantities of interest
    - Flow field from sensors at a few locations
    - Unknown PDE coefficients from data
  - We will look at finding unknown coefficients for a Lorentz system

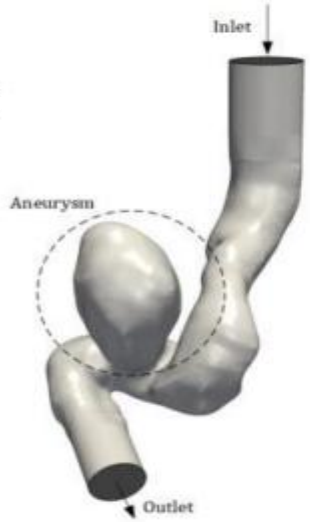


Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

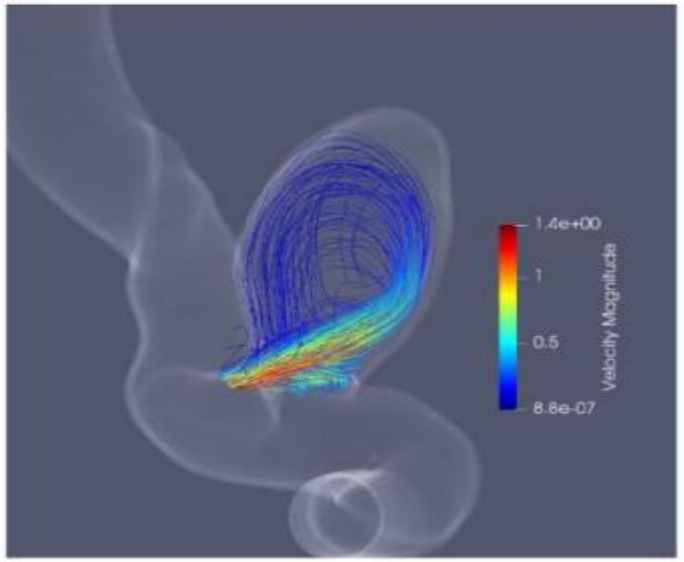
Raissi et al 2019

3. Applications of PINNs Forward Problems

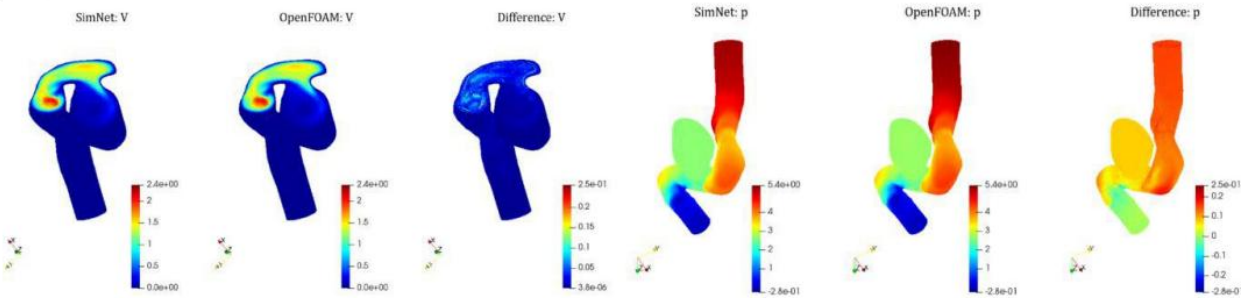
- Optimize PDE over auxiliary variables
  - FPGA design optimization of heatsink geometric configurations (Hennigh et al 2021)
- Simulations over very complex geometries
  - Brain aneurysm blood flow (Hennigh et al 2021)
  - Use transfer learning to reduce training time



(a) Geometry



(b) Streamlines



(c) Velocity magnitude comparison

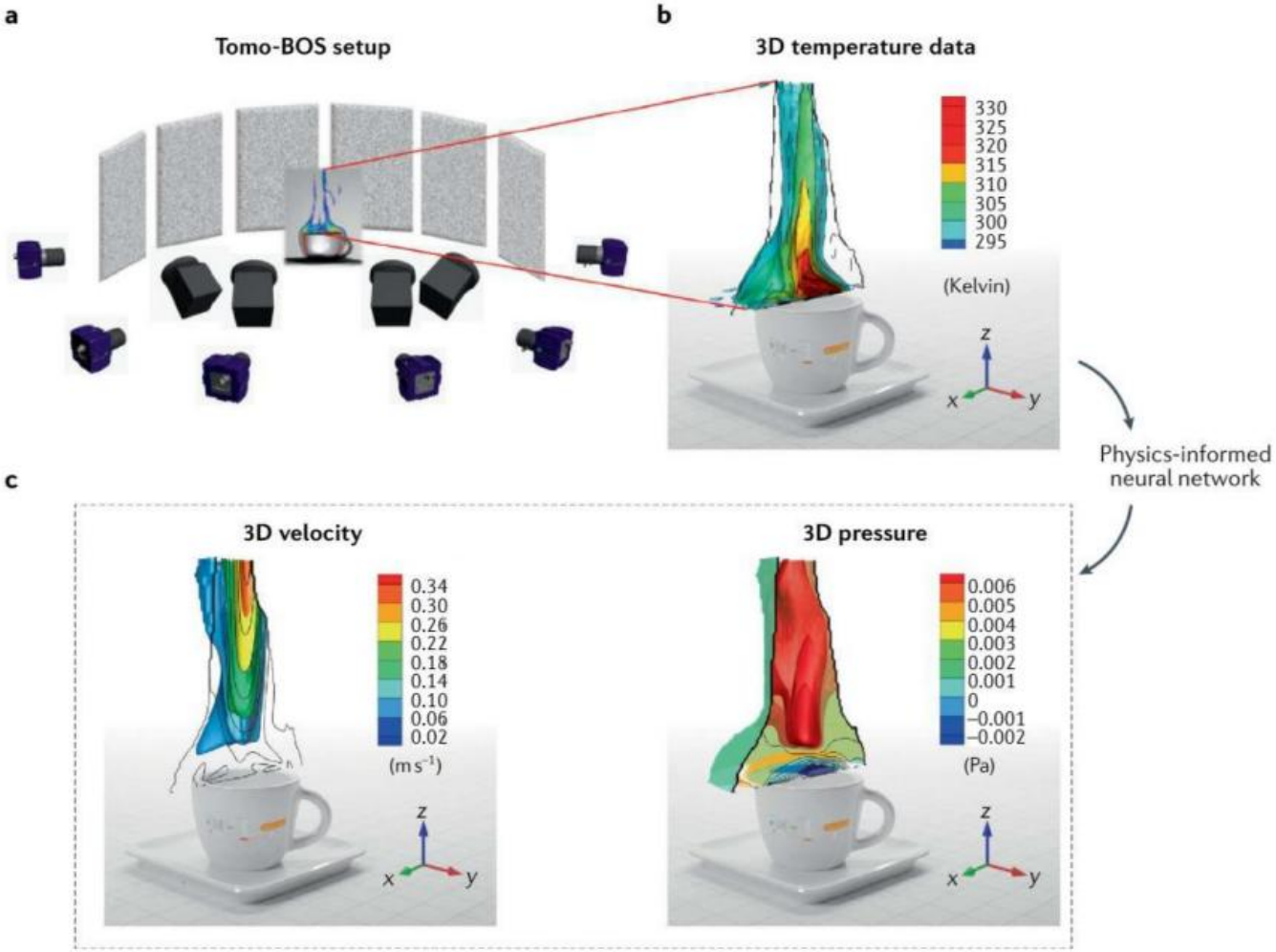
(d) Pressure comparison

Hennigh et al 2021

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Applications of PINNs Inverse Problems

- Reconstruct fields from limited sensor data in ill-posed problems
  - Construct fluid flow from a coffee cup (Cai et al. 2021)
  - Used temperature measurements to construct velocity and pressure data
- Analysis of scientific experiments
  - Well understood models
  - Controlled environments



Korniadakis et al 2021

PINN Software

- **Deepxde (https://github.com/lululxvi/deepxde) → Will use deepxde for our tutorials**
- NVIDIA Modulus/SimNet (Modulus | NVIDIA Developer)
- SciANN (https://github.com/sciann/sciann)
- Elvet (https://gitlab.com/elvet/elvet)
- TensorDiffEq (https://github.com/tensordiffeq/TensorDiffEq)
- NeuroDiffEq (https://github.com/analysiscenter/pydens)
- NeuralPDE (https://github.com/SciML/NeuralPDE.jl)
- Universal Differential Equations for Scientific Machine Learning (https://github.com/ChrisRackauckas/universal_differential_equations)
- IDRLnet (https://github.com/idrl-lab/idrlnet)

Limitations of PINNs

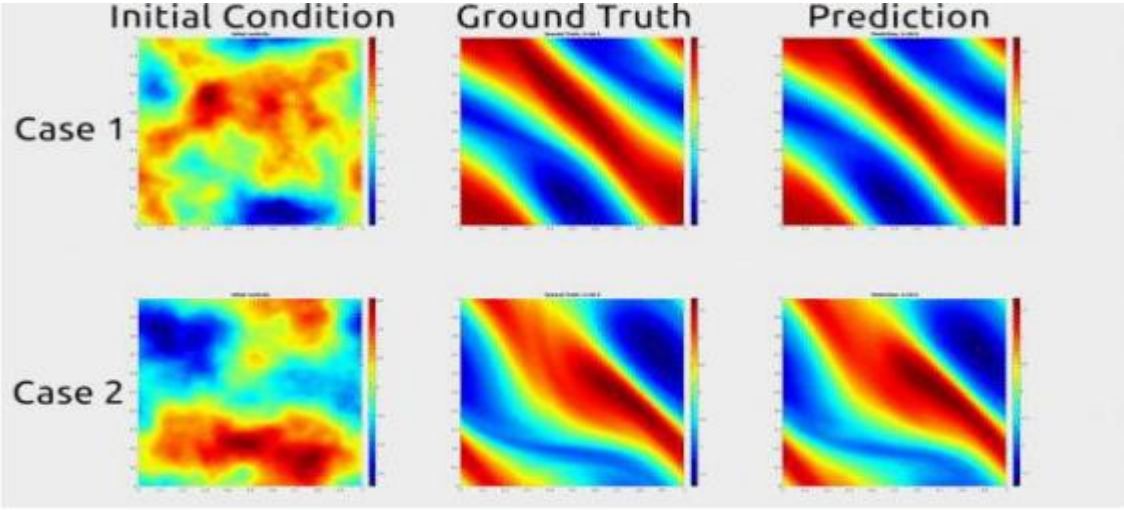
- Only trained for a single set of ICs/BCs/source terms → need to retrain for each new configuration
- Pure PINNs make poor surrogate models
- Will look at operator networks for solving PDEs with variable input fields

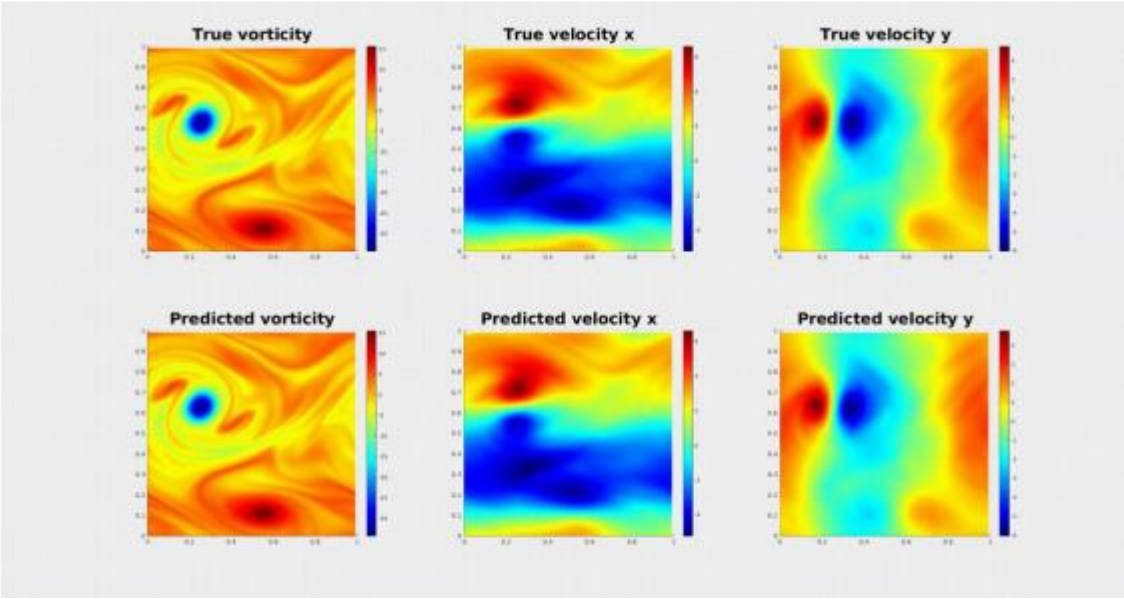
4. PINNs Exercises

- https://github.com/shawnrosofsky/HAL-Physics-Informed-AI-Tutorial

Operator Networks

- Learn output field for a given input field
- Can learn variable ICs, BCs, and/or source terms
- Need to generate data for many input fields
- May use physics information to improve performance
- Examples
  - DeepONets (Lu et al 2021)
  - Physics Informed DeepONets (Wang et al 2021)
  - Graph Operator Networks (Li et al 2020)
  - Fourier Operator Networks (Li et al 2020)
  - PINOs (Li et al 2021)

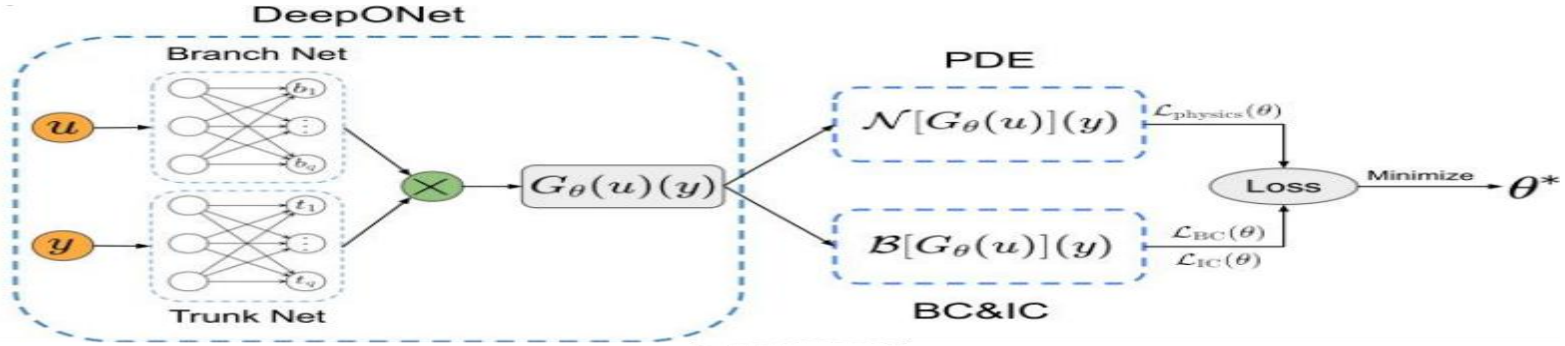




Neural Operator (zongyi-li.github.io)

5. Physics Informed DeepONets

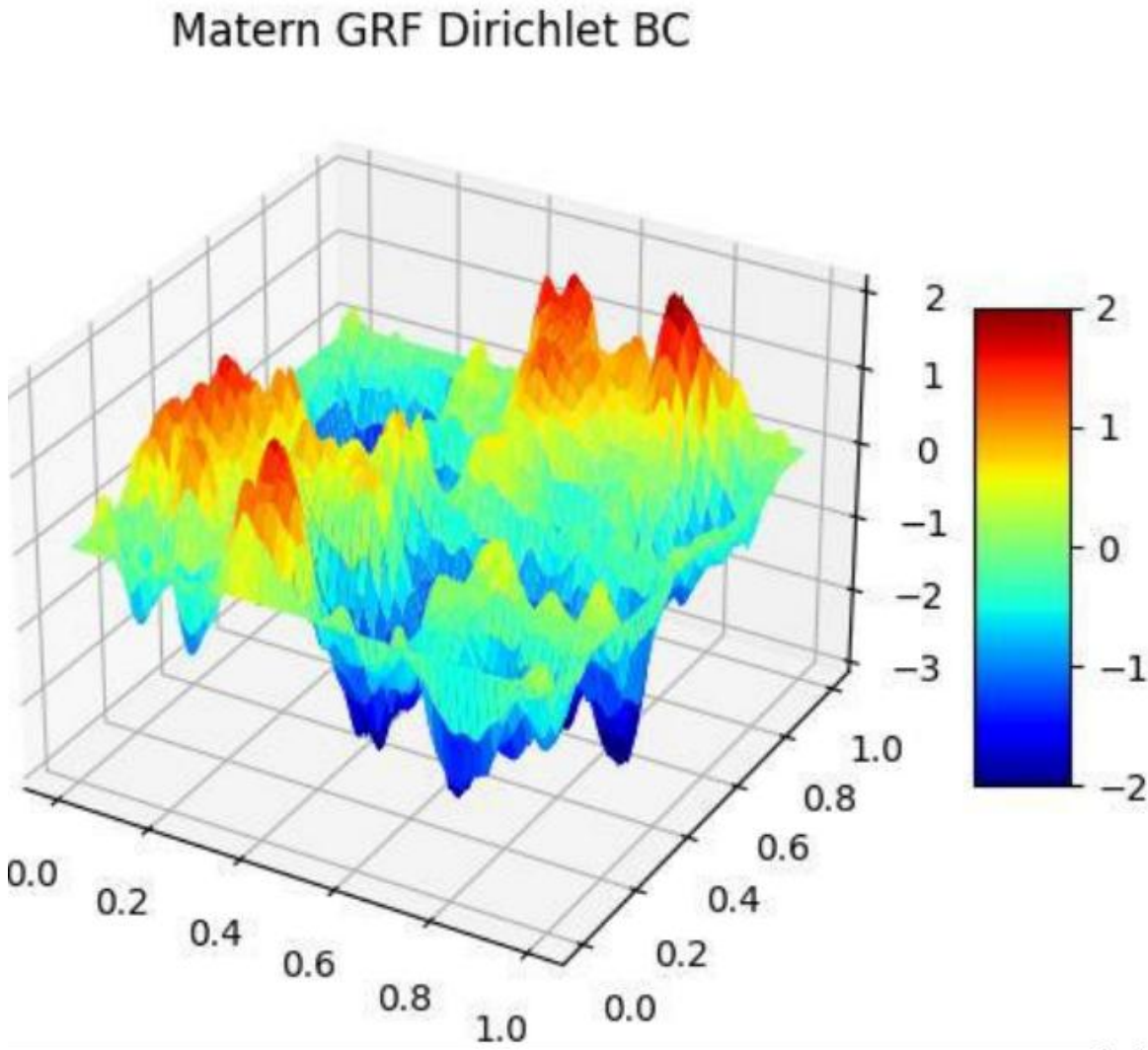
- DeepONets can generalize PDE solutions (Lu et al 2021)
  - Input field u → Initial conditions, source terms, and/or boundary conditions
  - Input coordinate y → space and time
  - Output operator $G(u)(y)$ → PDE solution
  - Difference between data s and operator $G\ (u)\ (y)$ is our loss $\mathscr{L}_{data}$
- Physics informed DeepONets improve performance with less data (Wang et al 2021)
  - Incorporate PDE into loss $\mathscr{L}_{physics}$
  - Incorporate ICs into loss $\mathscr{L}_{IC}$
  - Incorporate BCs into loss $\mathscr{L}_{BC}$



Wang et al 2021

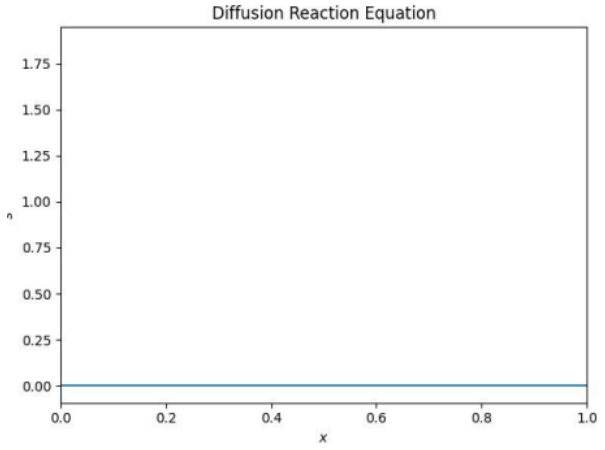
Training Physics Informed DeepONets

- Generate u using Gaussian random fields (GRF)
  - Use RBF or Matérn kernel to obtain spatially correlated random data
  - Apply length scale l associated with typical spatial deviations
  - Expand in Fourier components to obey boundary conditions
- Run simulations for each u to generate training data
- Sample the solution space during training

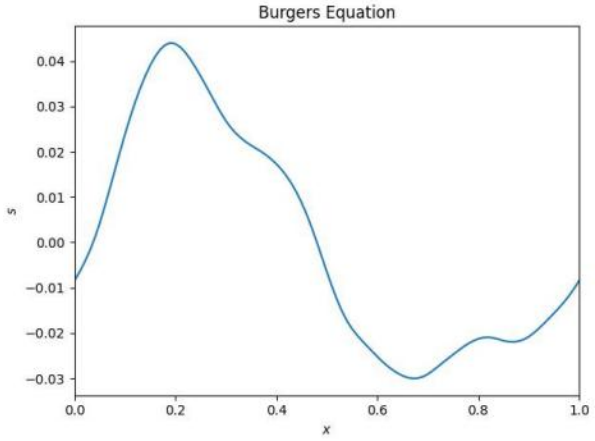


Physics Informed DeepONet Tests

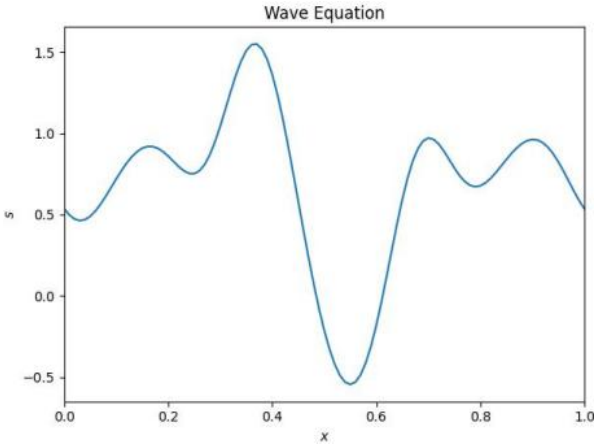
- 1D Diffusion Reaction Equation
  - $\partial_t s = D\,\partial_{xx} s + k\,s^2 + u(x)$
  - u is a source term
  - Homogenous Dirichlet BC
  - Zero IC → $s\,(x, 0) = 0$
  - $k = D = 0.01$



- 1D Viscous Burgers Equation
  - $\partial_t s + s\,\partial_x s - \nu\,\partial^2_{xx} s = 0$
  - u is the IC
  - Periodic BC
  - $\nu = 0.01$

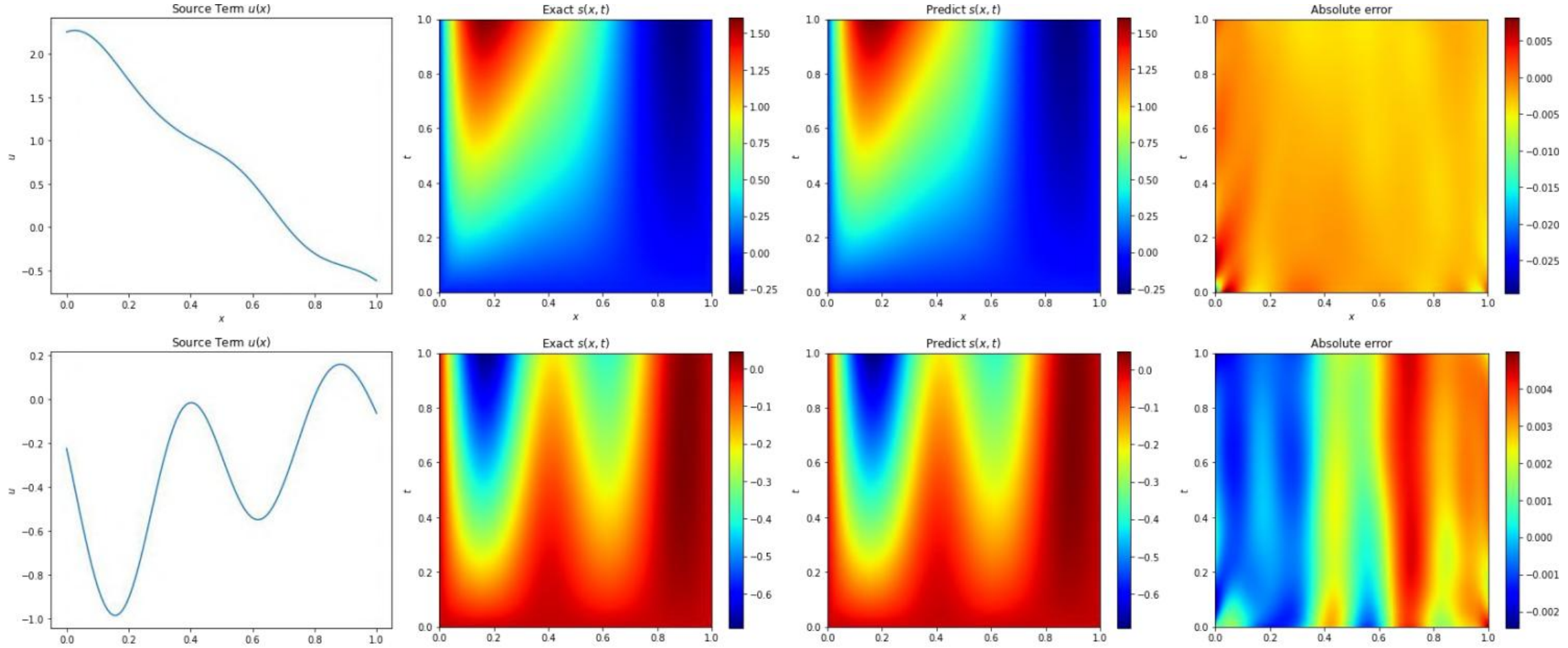


- 1D Wave Equation
  - $\partial_{tt} s - c^2 \partial^2_{xx} s = 0$
  - u is the IC
  - Periodic BC
  - $c = 1$



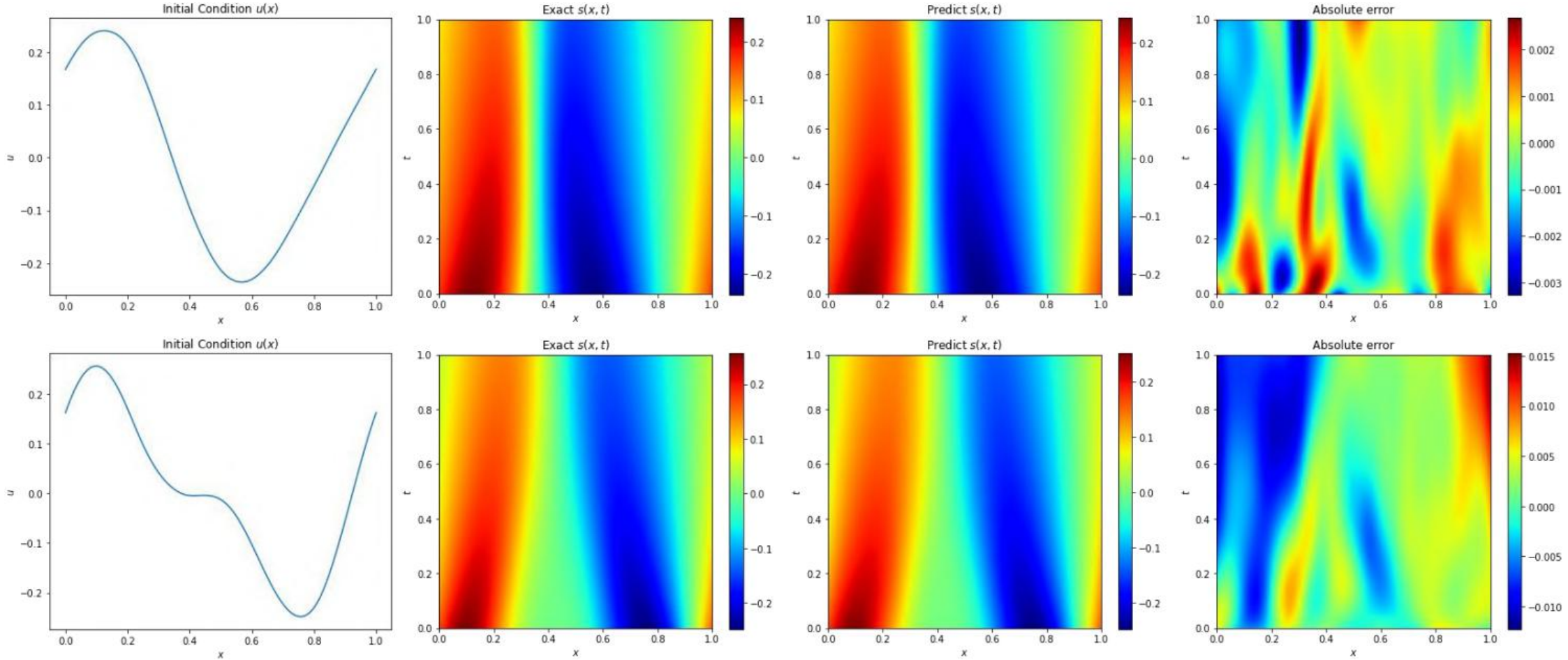
Diffusion Reaction Equation Results on Test Data:

$$\partial_t s = D\,\partial_{xx} s + k\,s^2 + u(x)$$



Viscous Burgers Equation Results on Test Data

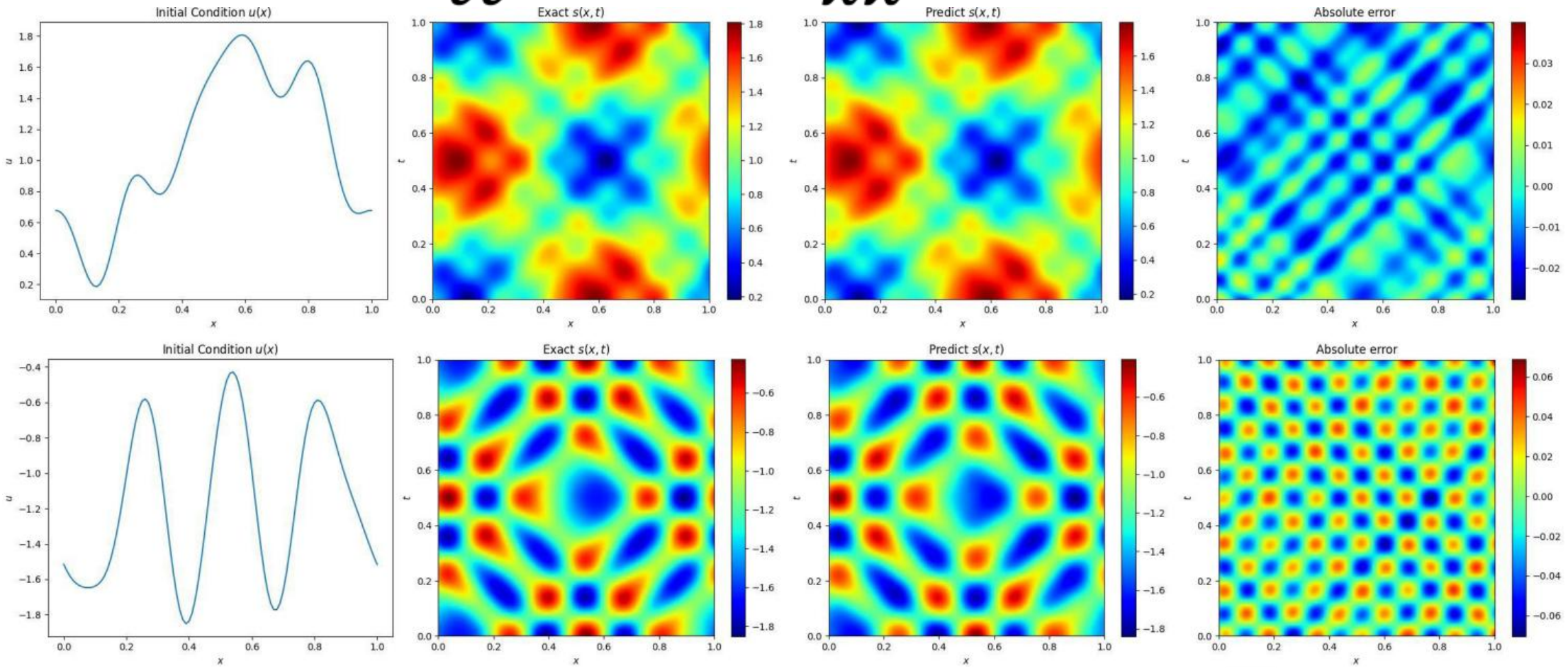
$$\partial_t s + s\,\partial_x s - \nu\,\partial_{xx}^2 s = 0$$



UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Wave Equation Results on Test Data

$$\partial_{tt} s - c^2 \partial^2_{xx} s = 0$$



6. Physics Informed DeepONet Exercises

- https://github.com/shawnrosofsky/HAL-Physics-Informed-AI-Tutorial

Chapter 4. Introduction To Reinforcement Learning

Jeong-woo Han, PhD
Mechanical & Mechatronics Engineering

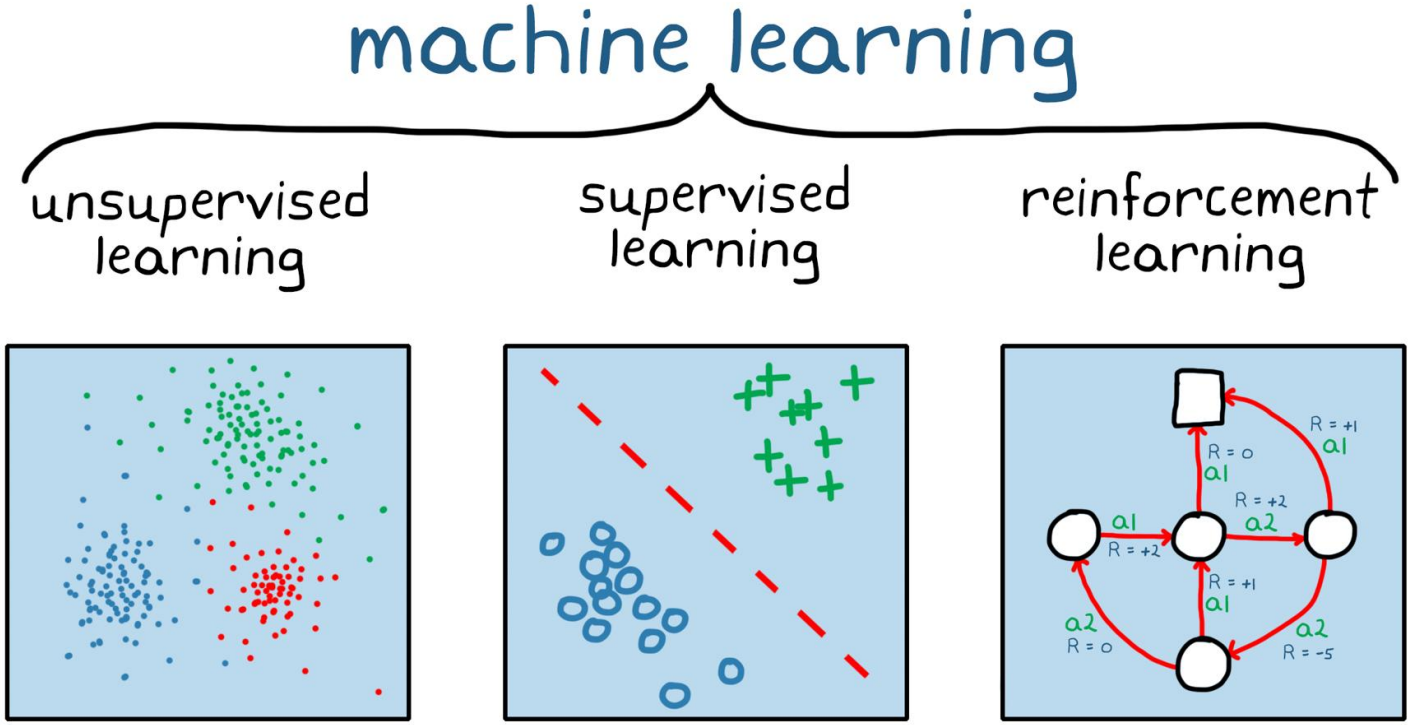
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Outlines

- Introduction and Fundamentals of RL
- Value-based Methods
- Deep Reinforcement Learning

INTRODUCTION AND FUNDAMENTALS OF RL

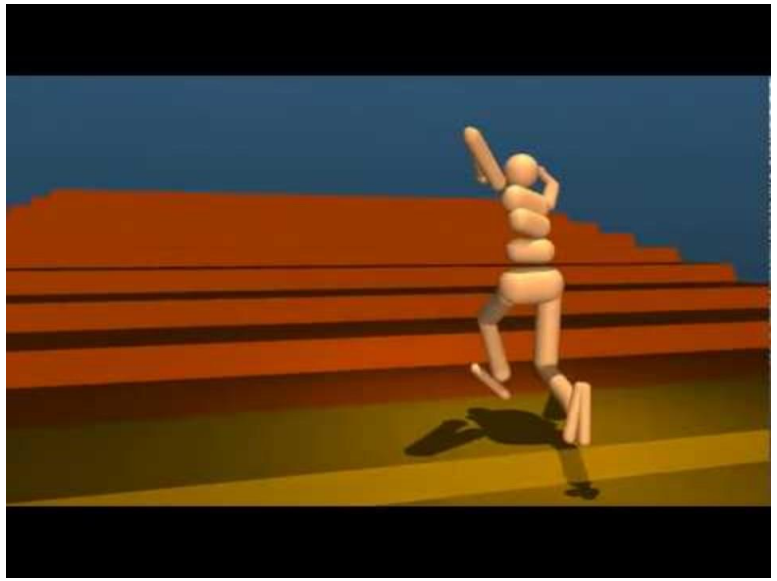
What is Reinforcement Learning?



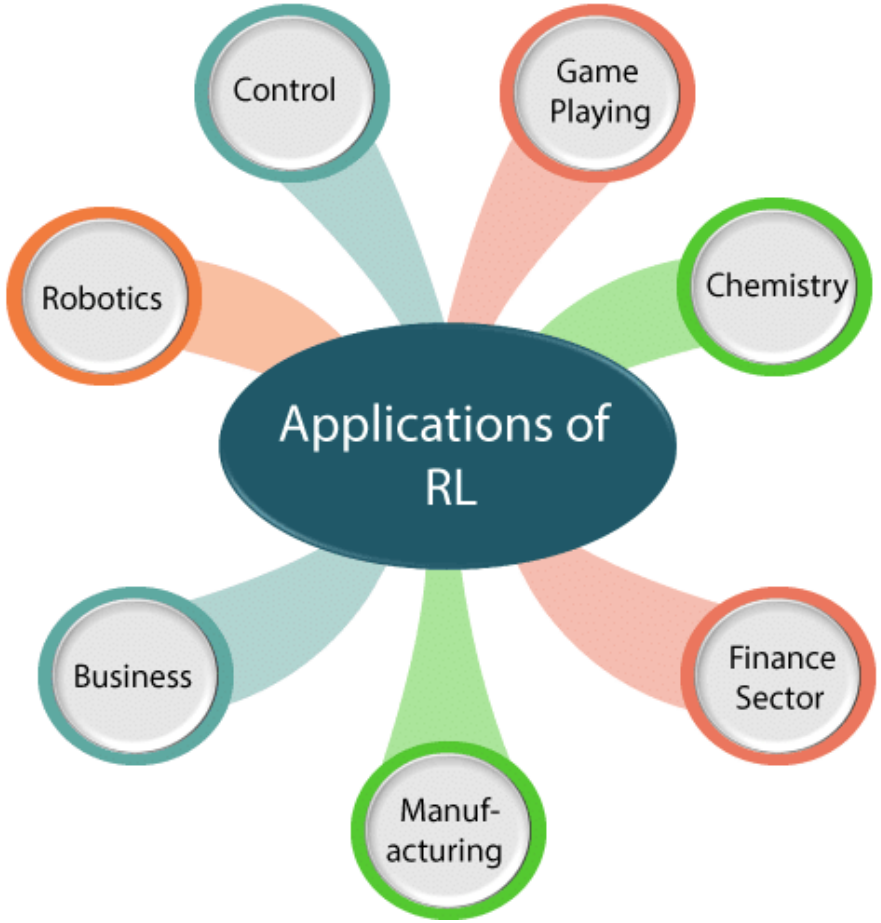
- Reinforcement Learning?
  a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative reward. The basic components of reinforcement learning include:

Applications of RL

- Marketing
- Broadcast Journalism
- Robotics
- Controls
- Healthcare
- Gaming
- Etc.

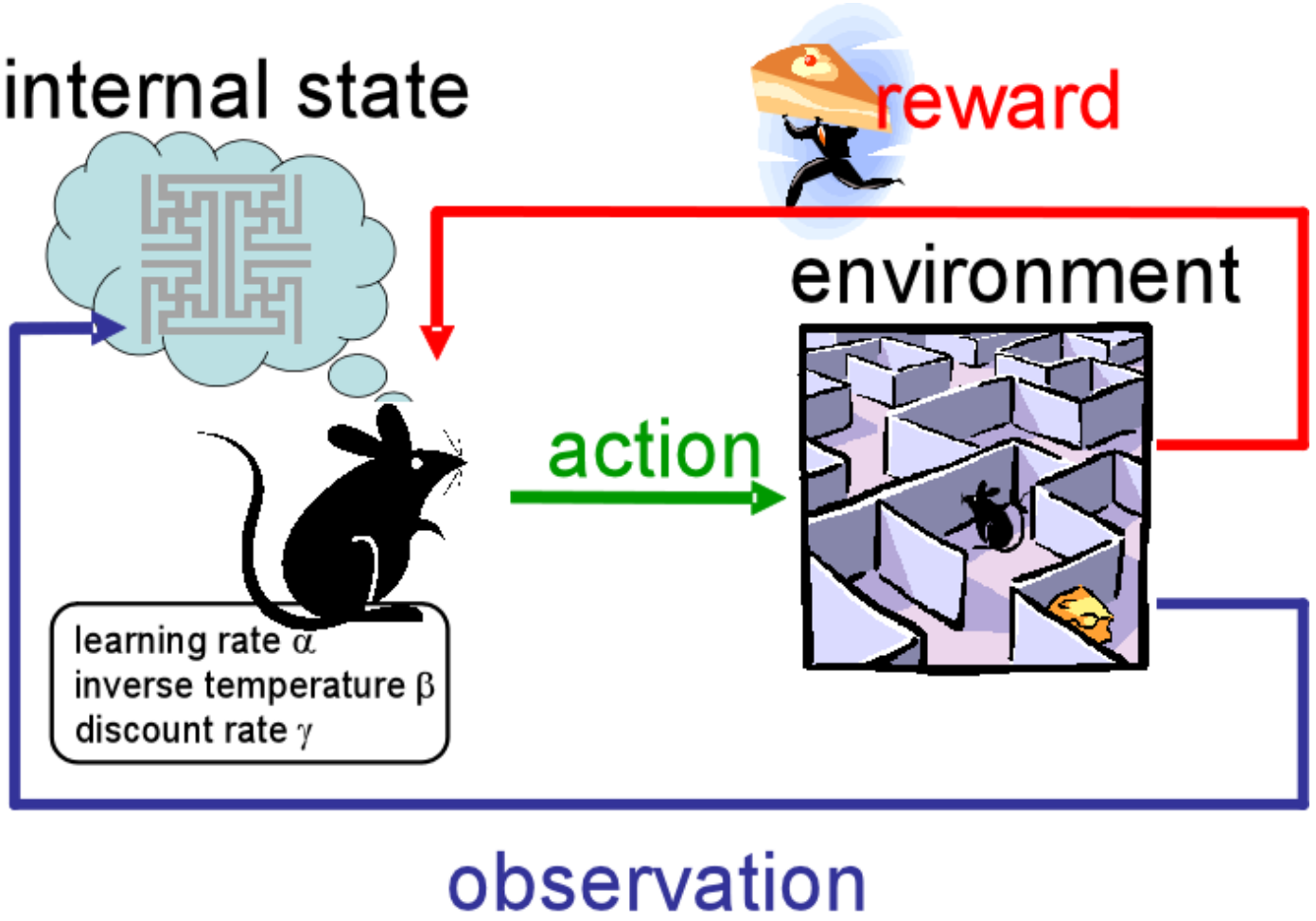


https://www.youtube.com/watch?v=hx_bgoTF7bs



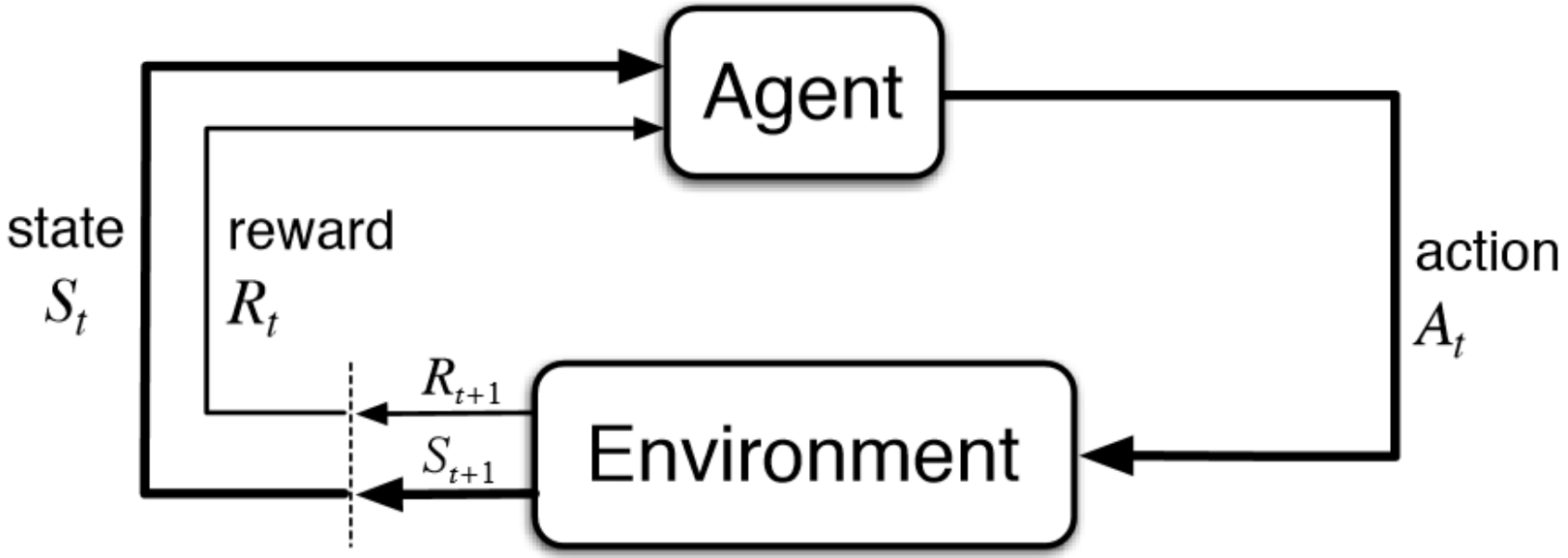
Key Concepts

- Agent
- Environment
- State
- Action
- Reward



Key Concepts

- Agent
- Environment
- State
- Action
- Reward

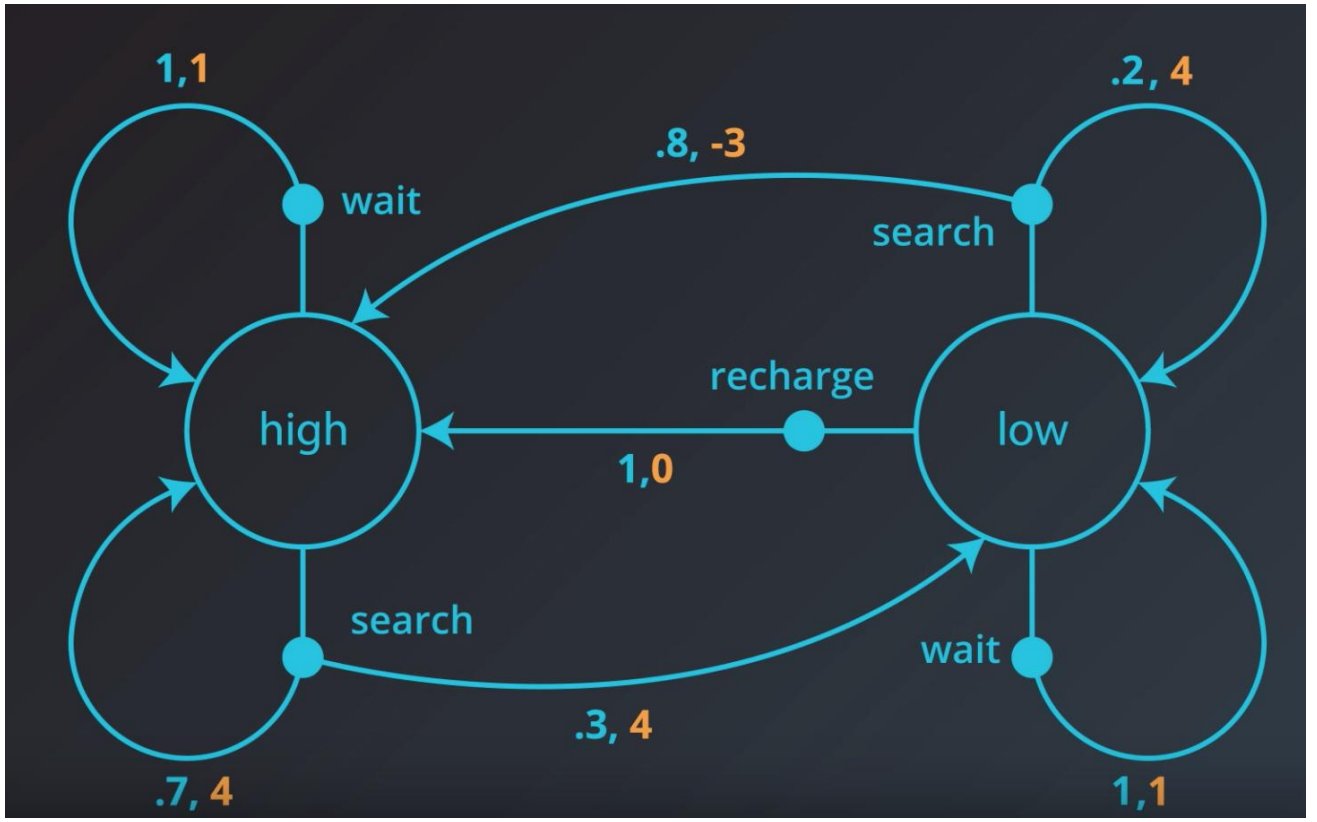


What is MDP?

- A (finite) Markov Decision Process (MDP)
  - a (finite) set of states, $\mathcal{S}$ (known)
  - a (finite) set of actions, $\mathcal{A}$(known)
  - a (finite) set of rewards, $\mathcal{R}$
  - the one-step dynamic of the environment (transition model)

$$p(s', r|s, a) = p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$$

  - A discount rate $\gamma \in [0,1]$
- Markov Property



Recycling Robot MDP

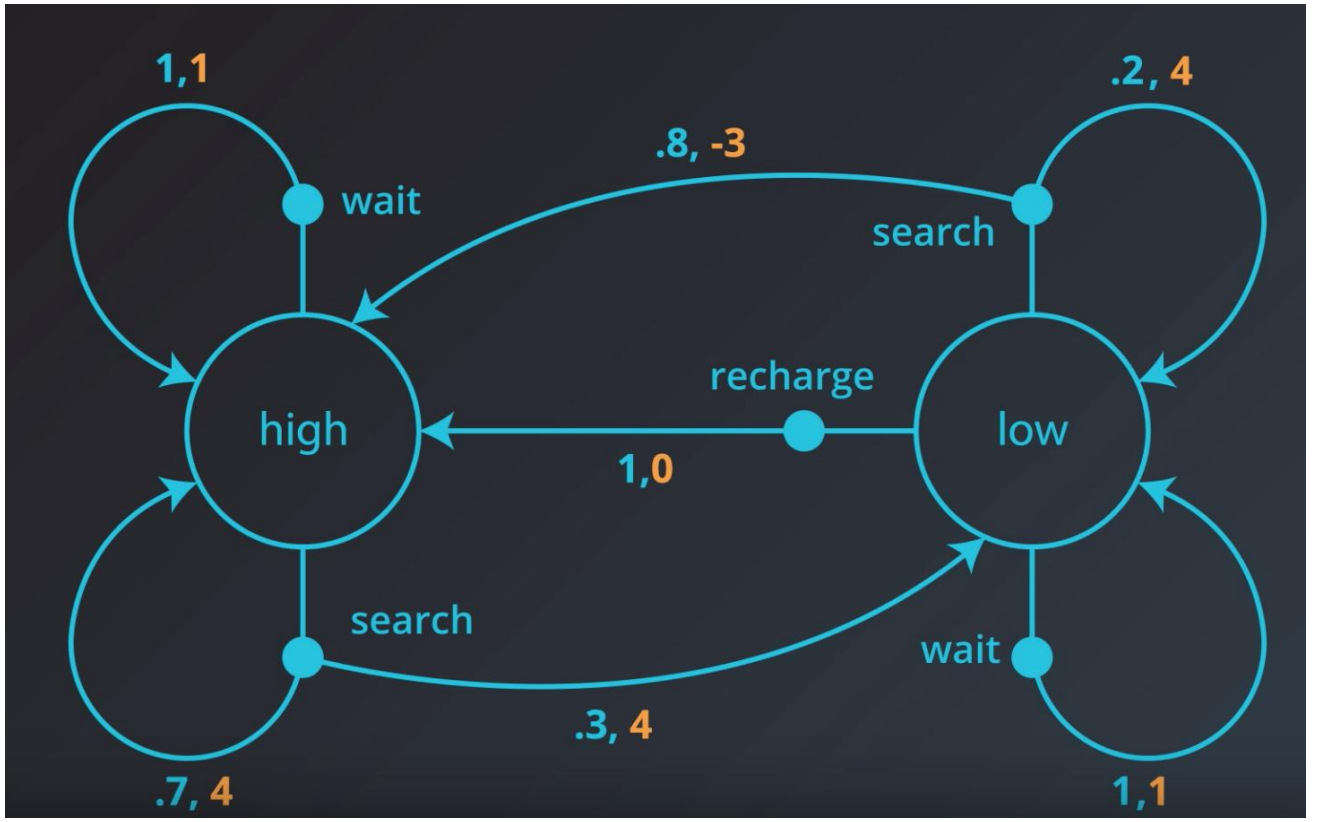
What is MDP?

- Deterministic Policy

$$\pi(low) = recharge$$
$$\pi(high) = search$$

- Stochastic Policy

$$\pi(recharge|low) = 0.5$$
$$\pi(wait|low) = 0.4$$
$$\pi(search|low) = 0.1$$
$$\pi(search|high) = 0.9$$
$$\pi(wait|high) = 0.1$$



Recycling Robot MDP

Bellman Equations

- **Return:**
  cumulative rewards with discount factor γ until the end of episode from a given state s

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ \ldots + \gamma^{N-t} R_{t+N}$$
$$= \sum_{k=t}^{N} \gamma^{k-t} R_k$$

- **Expected Return:**
  The mean of returns for all possible episodes from a given state s, defined as state-value

$$V(s) \triangleq \mathbb{E}[G_t]$$

- **Bellman Equation:**
  Recall return and state-value at a state at time at t, s_t

$$G_t = R_t + \gamma\left(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots + \gamma^{N-(t+1)} R_{t+N}\right)$$
$$= R_t + \gamma \sum_{k=t+1}^{N} \gamma^{k-(t+1)} R_k = R_t + \gamma G_{t+1}$$
$$V(s_t) = \mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}] = \mathbb{E}[R_{t+1} + \gamma V(s_{t+1})]$$

Bellman Equations

- State-Value Function

$$V(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$$

value of any state s

the expected value of

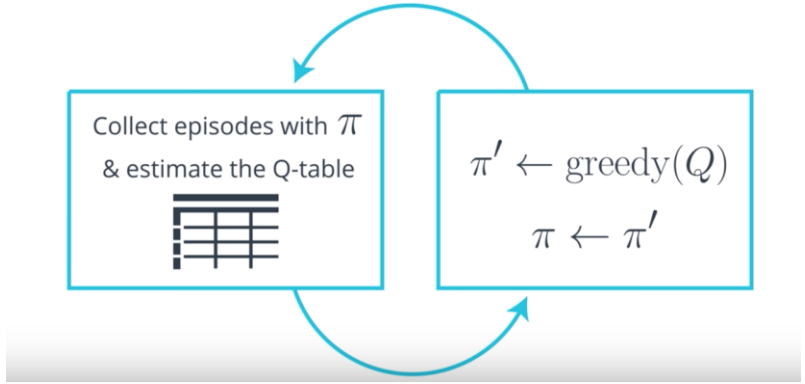
immediate reward + discounted value of the state that follows

- Action-Value Function

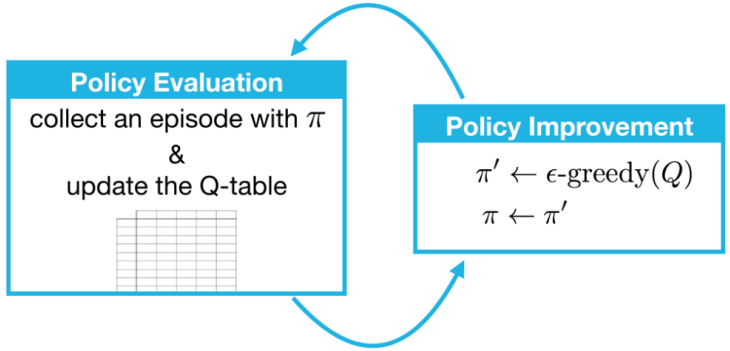
$$Q(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

How to find optimal policy

- Greedy Policy
  - ➢ always selects the greedy action

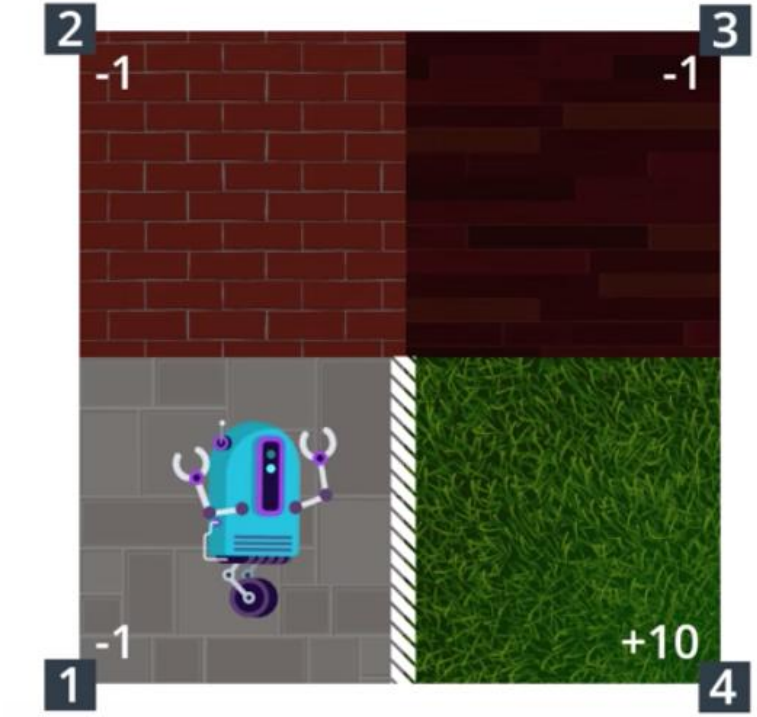


- Epsilon-Greedy Policy
  - ➢ most likely selects the greedy action



How to find optimal policy

- Grid World Example



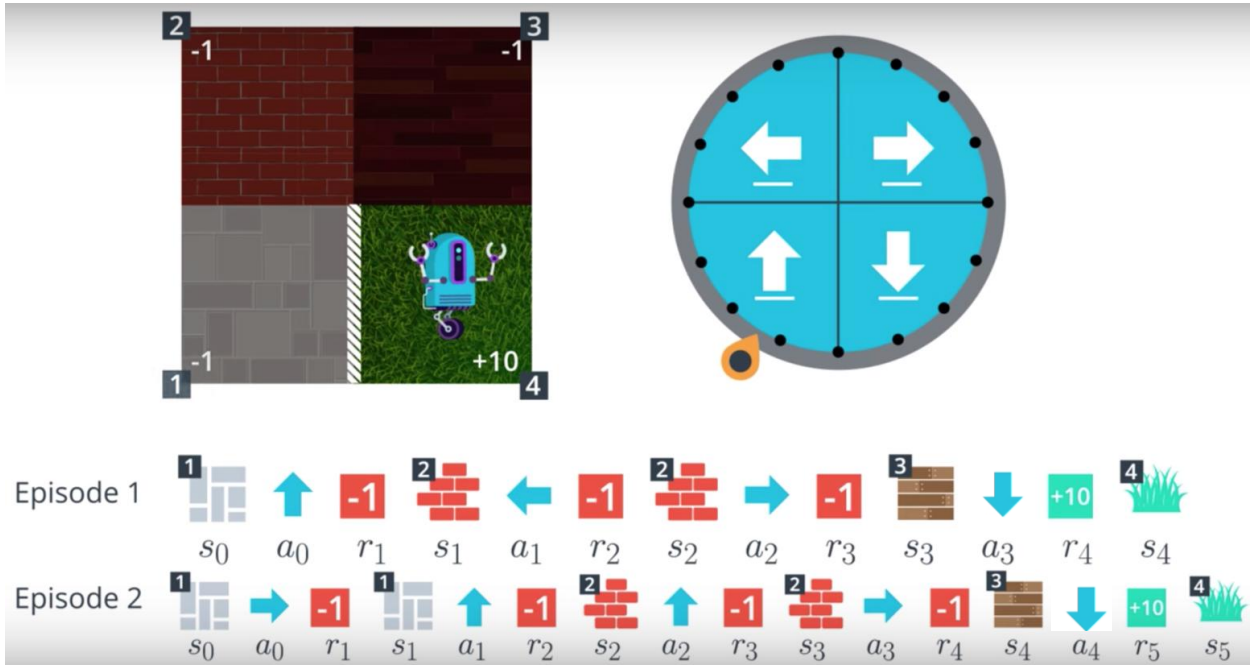
Action Space

Discount Factor

$\gamma = 1$

How to find optimal policy

- Grid World Example



Q-Table

1	+7	+6	+5	+6
2	+8	+7	+8	+9
3	+10	+8	+9	+9

Which action is best for each state?

$\pi'($ 1 $) =$ ↑

$\pi'($ 2 $) =$ →

$\pi'($ 3 $) =$ ↓

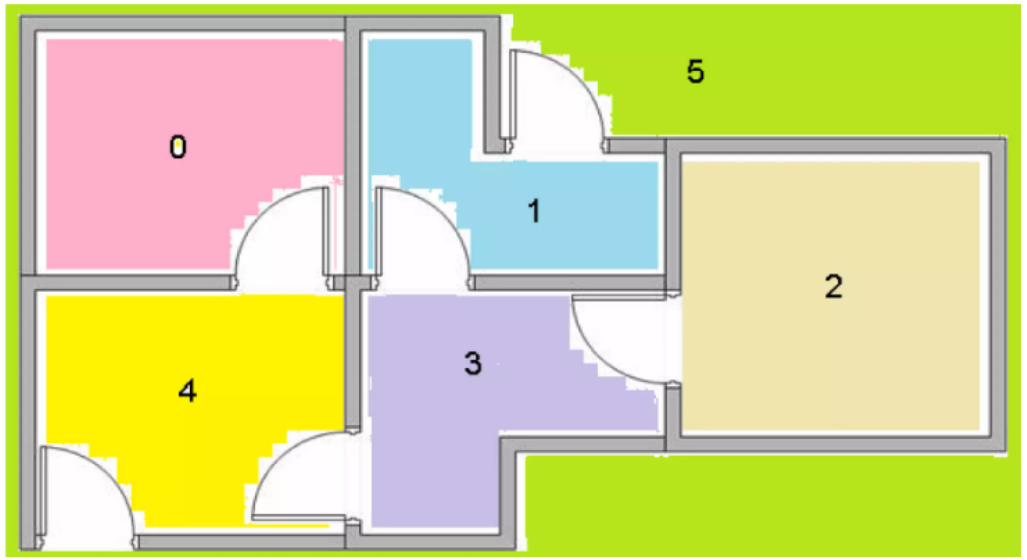
$\pi' \leftarrow \text{greedy}(Q)$

Q-Learning

- Action-value function $Q(s_t, a_t)$
- Target is to maximize Q: $a(s) = \underset{a}{\text{argmax}}\, Q_\pi(s, a)$
- Q-Learning update rule

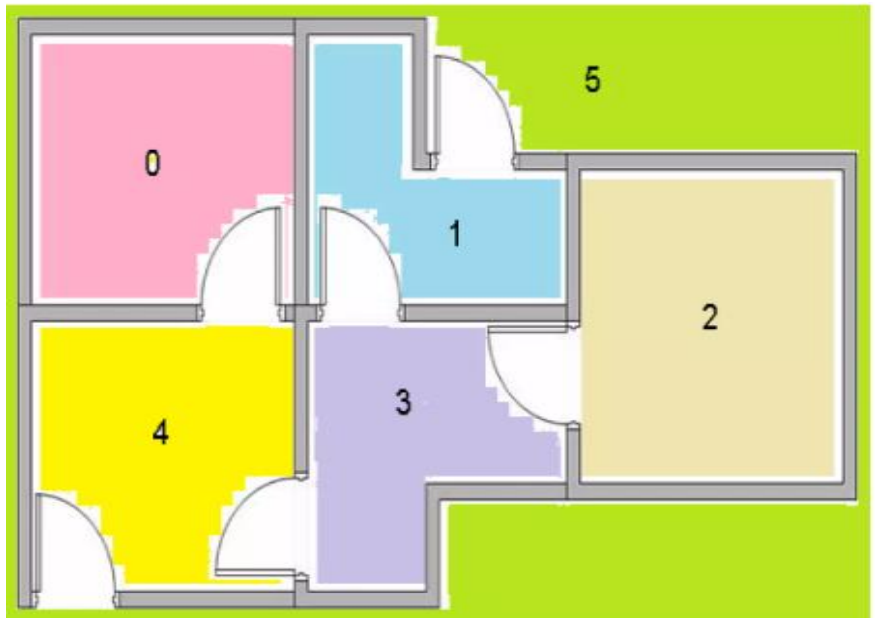
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

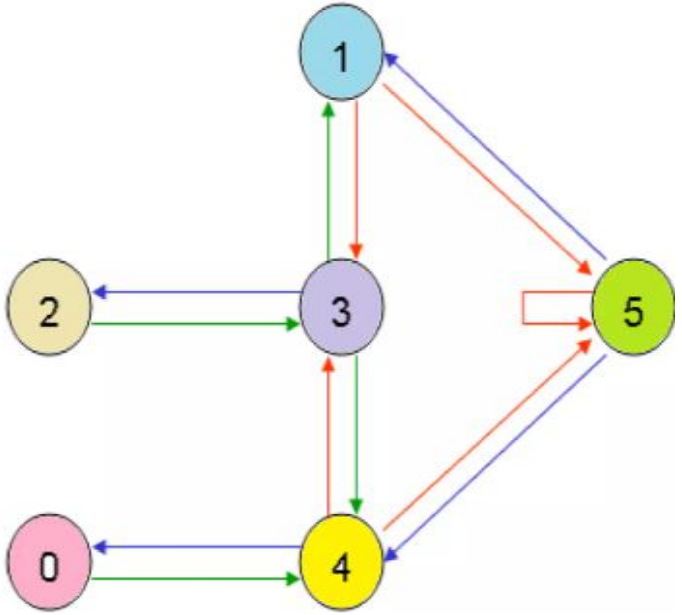
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q
    Take action a, observe r, s'
    Update
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$;
  Until s is terminal



Q-Learning

- Building with 5 rooms, numbered from 0 to 4, and outside numbered 5
- Problem: Agent is placed in any of the rooms (0~4) and needs to outside (5)

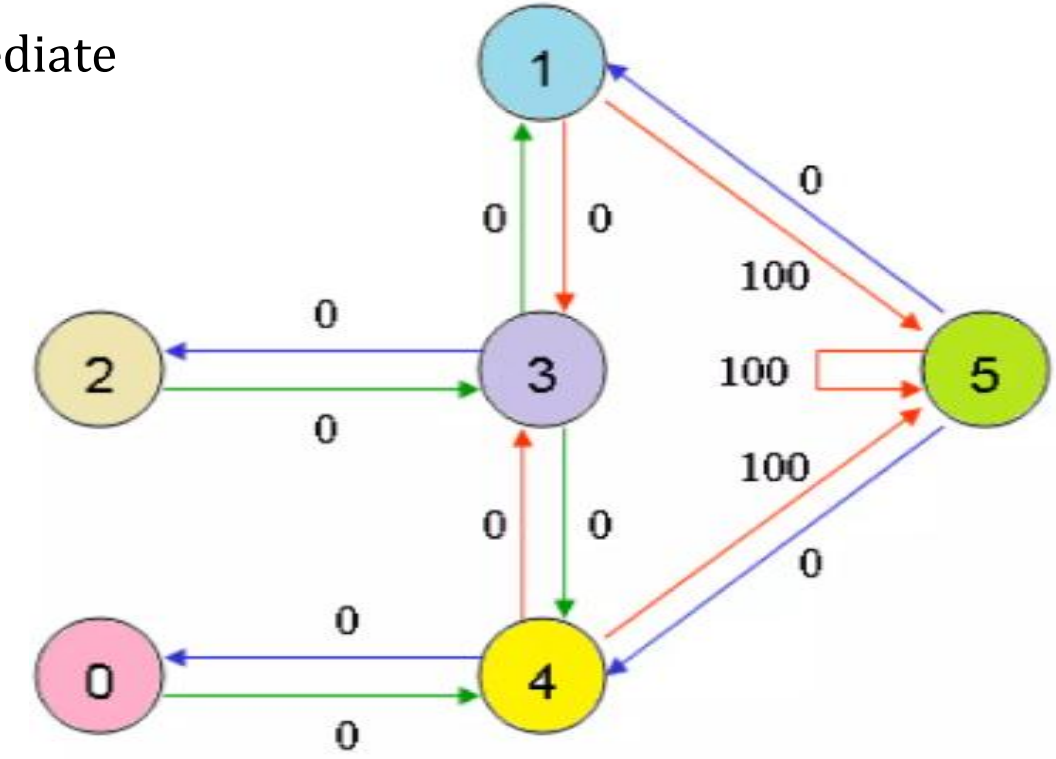




Goal state

Q-Learning

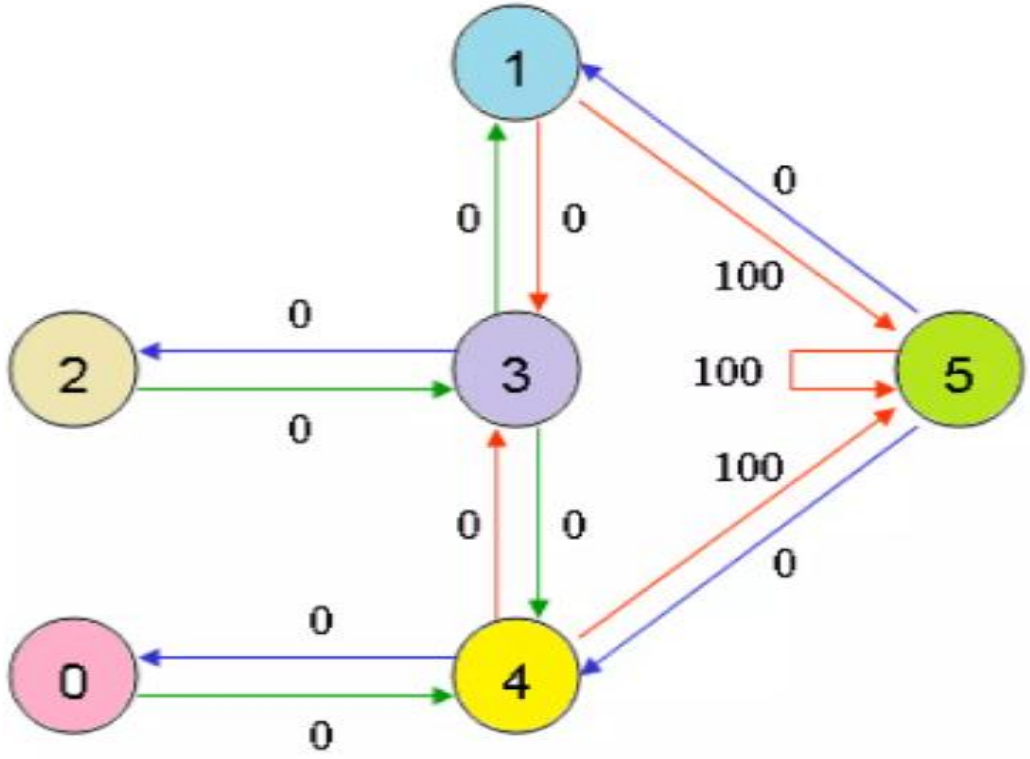
- Assign reward to each door
  - The doors leading to the outside (room 5) have immediate reward of **100**
  - Other doors are reward of **0**
  - Doors are two-way (e.g., 0 to 4 / 4 to 0)
  - Each edge contains an instant reward



Q-Learning

- Prepare rewards table R (matrix)

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



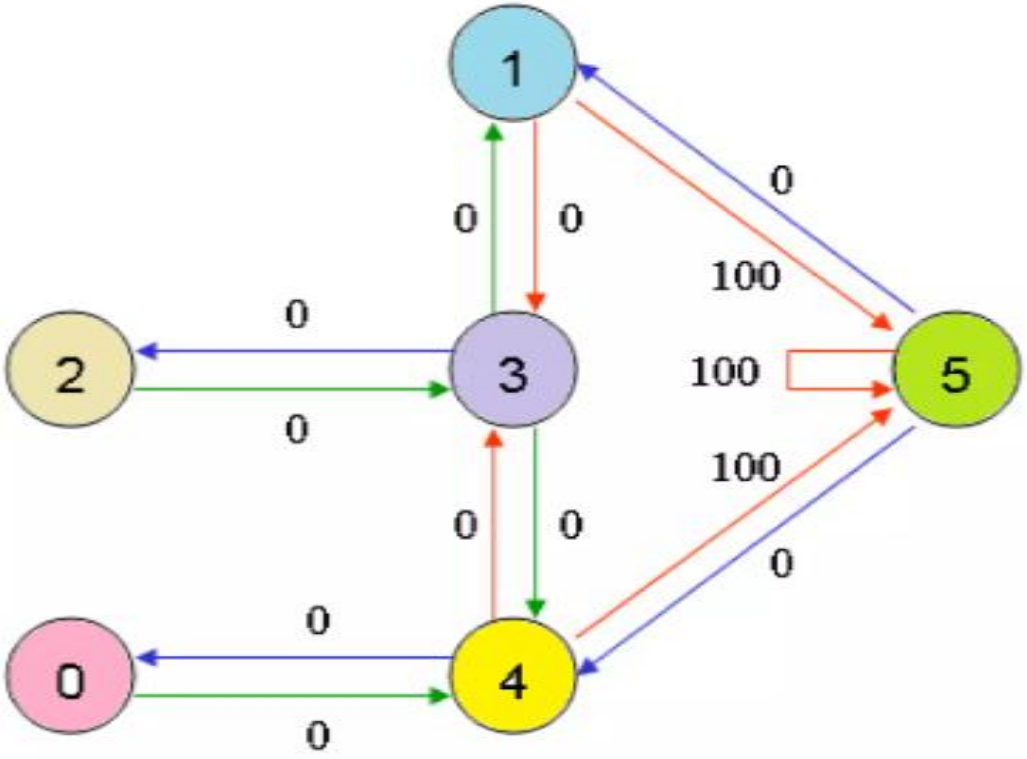
- -1 represents no edge

Q-Learning

- Prepare action-value Q (matrix)
- Q is the memory of the agent to store the experience information
- Row denotes the current state of the agent
- Column denotes the possible actions leading to the next state
- Compute Q matrix:

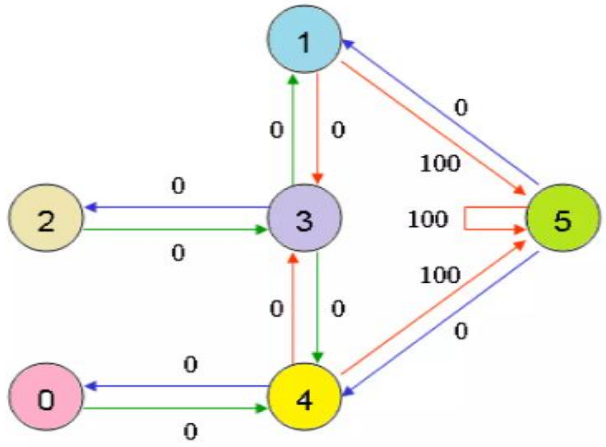
$$Q(S_t, A_t) = R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1})$$

- Future rewards are considered depending on γ



Q-Learning

- Example:
- $\gamma = 0.8$
- Initialize Q

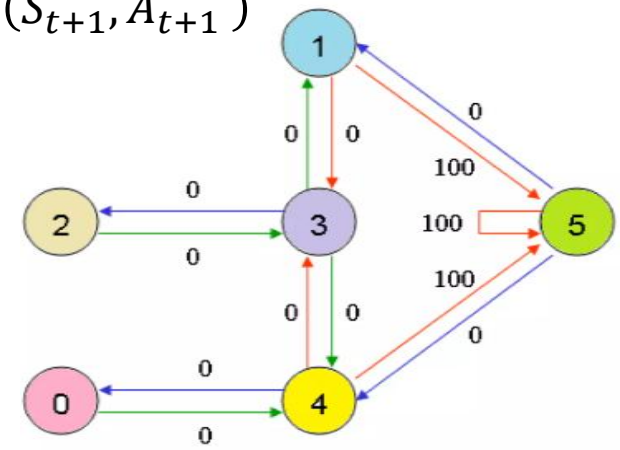


$$Q = \begin{array}{c} \\ \text{State } 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{c} \\ \text{State } 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Q-Learning

$$Q(S_t, A_t) = R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1})$$



- From state 1, agent can go to either state 3 or 5
- Let's choose 5
- From 5, compute $\max_{a'} Q(s_{t+1}, a')$ to go to next state with all possible actions

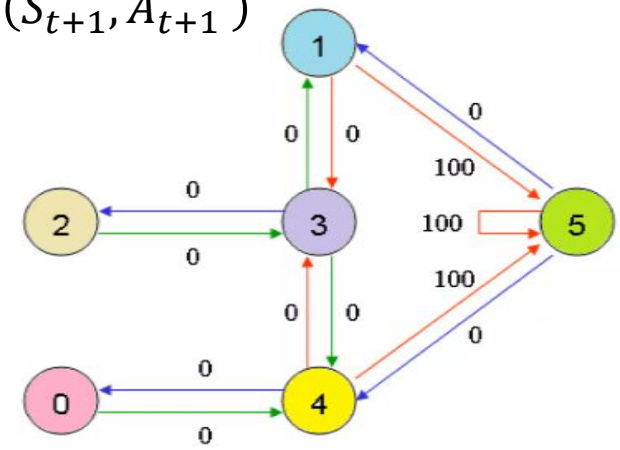
$$Q(1,5) = R(1,5) + 0.8 \max[Q(5,1), Q(5,4), Q(5,5)]$$
$$= 100 + 0.8 \max[0,0,0] = 100$$

- Update Q

$$Q = \begin{array}{c} \\ \text{State} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} \end{array} \begin{array}{c} \begin{matrix} \text{Action} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{100} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad R = \begin{array}{c} \\ \text{State} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} \end{array} \begin{array}{c} \begin{matrix} \text{Action} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Q-Learning

$$Q(S_t, A_t) = R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1})$$



- For next episode, choose next state 3 randomly that becomes current state
- State 3 contains 3 choices: room 1, 2, and 4. → Let's choose 1
- Compute $\max_{a'} Q(s_{t+1}, a')$ to go to next state with all possible actions

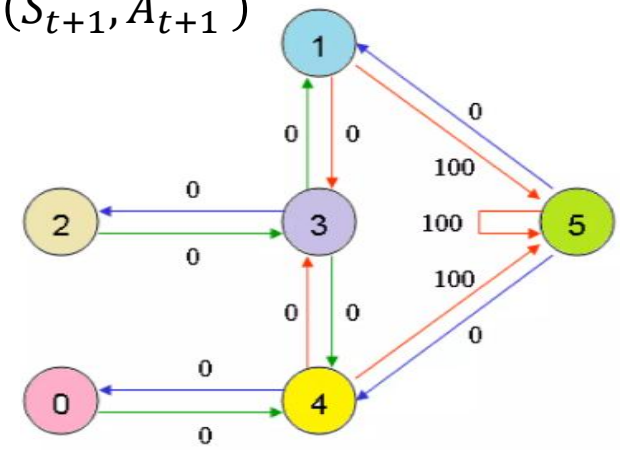
$$Q(3,1) = R(3,1) + 0.8 \max[Q(1,3), Q(1,5)]$$
$$= 0 + 0.8 \max[0,100] = 80$$

- Update Q

$$Q = \begin{array}{c} \\ \text{State } 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action } 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{80} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad R = \begin{array}{c} \\ \text{State } 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action } 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Q-Learning

$$Q(S_t, A_t) = R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1})$$



- State 1 now became current state
- State 1 contains 2 choices: room 3 and 5. → Let's choose 5
- Compute $\max_{a'} Q(s_{t+1}, a')$ to go to next state with all possible actions

$$Q(1,5) = R(1,5) + 0.8 \max[Q(5,1), Q(5,4), Q(5,5)]$$
$$= 100 + 0.8 \max[0,0,0] = 100$$

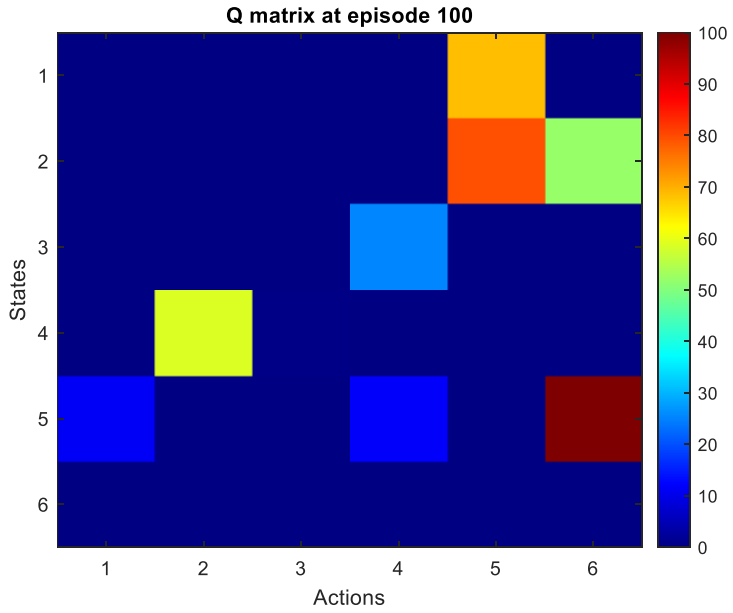
- Q remains the same due to Q(1,5) is already fed into the agent

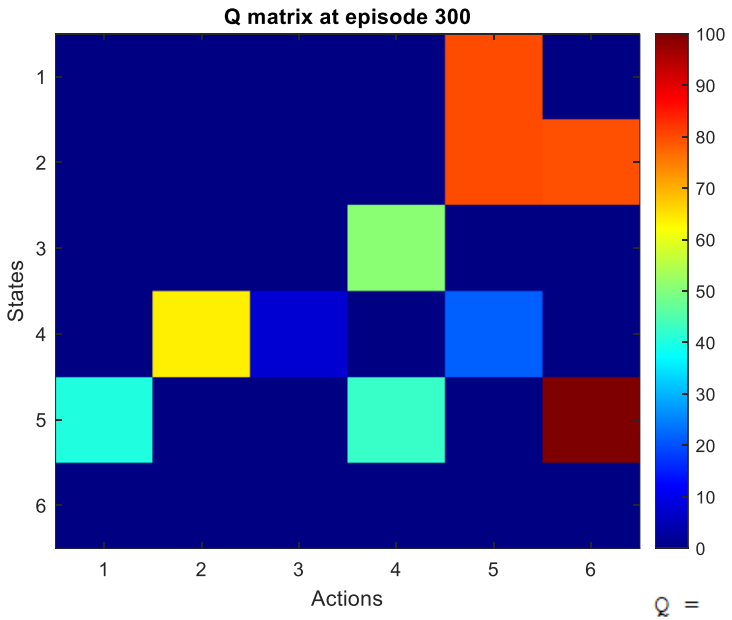
$$Q = \begin{array}{c} \\ \text{State} \\ \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action} & & & & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

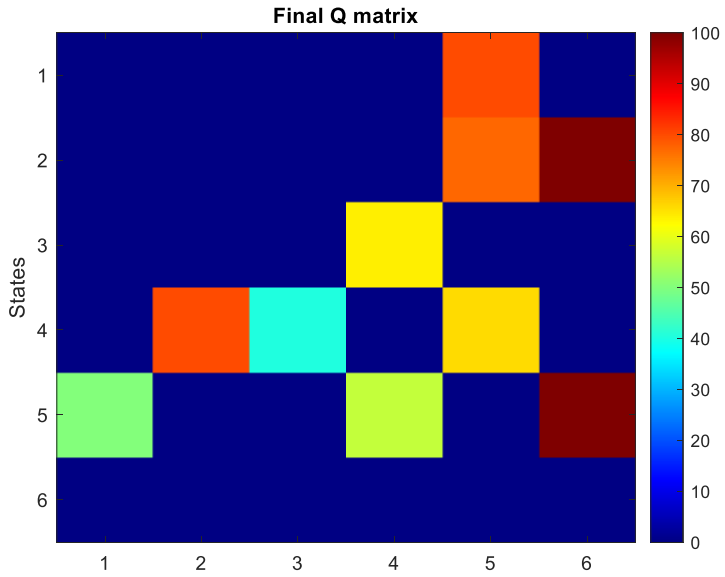
$$R = \begin{array}{c} \\ \text{State} \\ \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action} & & & & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

Q-Learning

- Check out the code on NAS







Q =

-0.1000	-0.1000	-0.1000	-0.0732	80.0000	0
-0.1000	-0.1000	-0.1000	-0.1000	80.0000	100.0000
-0.1000	-0.1000	-0.1000	64.0000	0	0
-0.1000	80.0000	25.7165	0	52.1057	0
55.7622	0	0	56.9458	0	100.0000
0	0	0	0	0	0

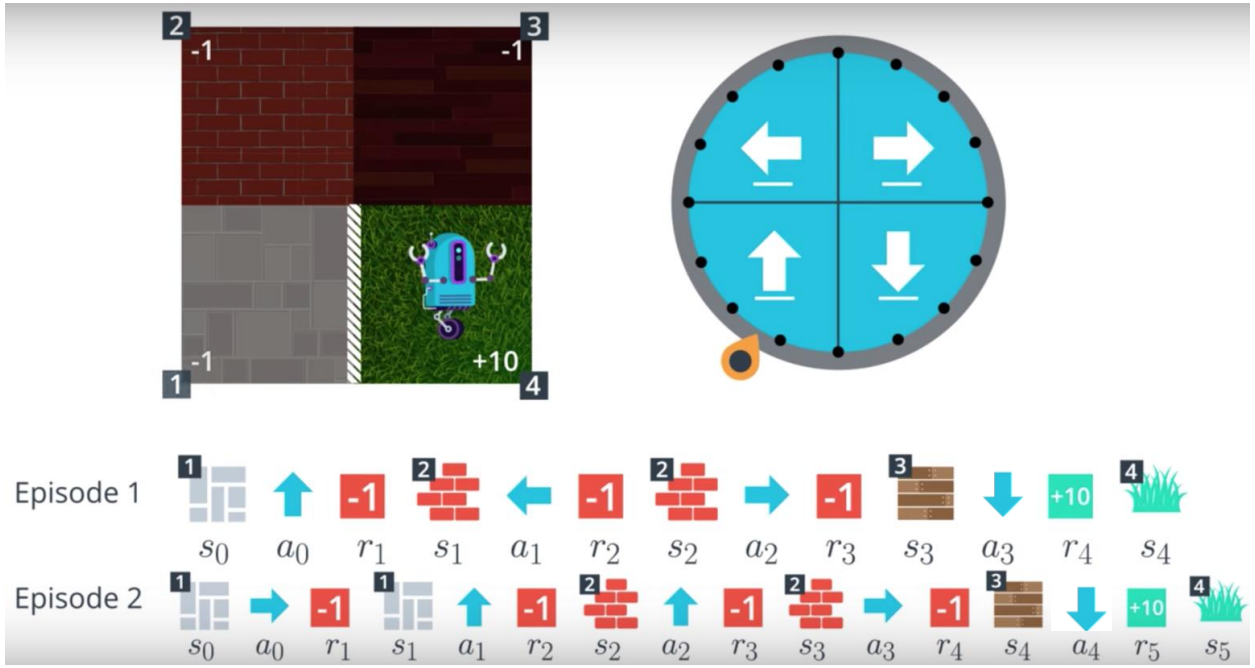
Q-Learning

- Some Remarks on Q-Learning
  - Model-free
    - Suitable for complex and unknown environments
  - Off-policy
    - It learns the value of the optimal policy independently of the agent's actions (so off-policy), which allows the use of exploratory actions during learning
  - Convergence
    - With sufficient exploration, a decaying learning rate, and a discount factor, Q-Learning is proven to converge to the optimal action-value function $Q^*(s, a)$

VALUE BASED METHODS

Monte Carlo Learning

- Let's go back to the earlier grid world example



Q-Table

1	+7	+6	+5	+6
2	+8	+7	+8	+9
3	+10	+8	+9	+9

Which action is best for each state?

$\pi'(1) =$ ↑

$\pi'(2) =$ →

$\pi'(3) =$ ↓

$\pi' \leftarrow \text{greedy}(Q)$

Monte Carlo Learning

- Recall Bellman Equation

$$V(s_t) = \mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}] = \mathbb{E}[R_{t+1} + \gamma V(s_{t+1})]$$

- Straight-forward way to get the expectation $\mathbb{E}[G_t]$
- Do a lot of experiment (law of large numbers) until the end of the episodes

$$\mathbb{E}[G_t] \approx \frac{1}{N}\sum_{k=1}^{N} G_t^{(k)}$$

- Instead of varying (increasing) number of N, we consider a learning rate α

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t \qquad \text{for state-value function}$$

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha G_t \qquad \text{for action-value function}$$

Temporal Difference (TD) Learning

- For MC, we need to wait until the episode ends to get G_t
  - Takes long time to converge
  - High variances
- Instead, we can assume the value of next state approximates the following return:

$$G_{t+1} \approx V(s_{t+1})$$

- With next reward (immediately available), G_t can be approximated by the sum of the next reward and its state-value with some discount:

$$G_t \approx R_{t+1} + \gamma V(S_{t+1})$$

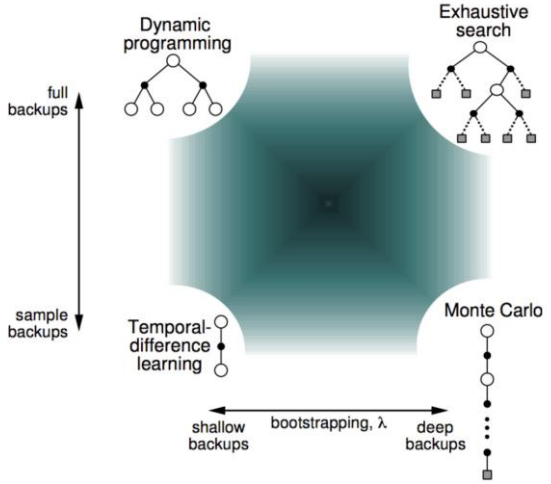
- Like MC, we can think of training with a learning rate α

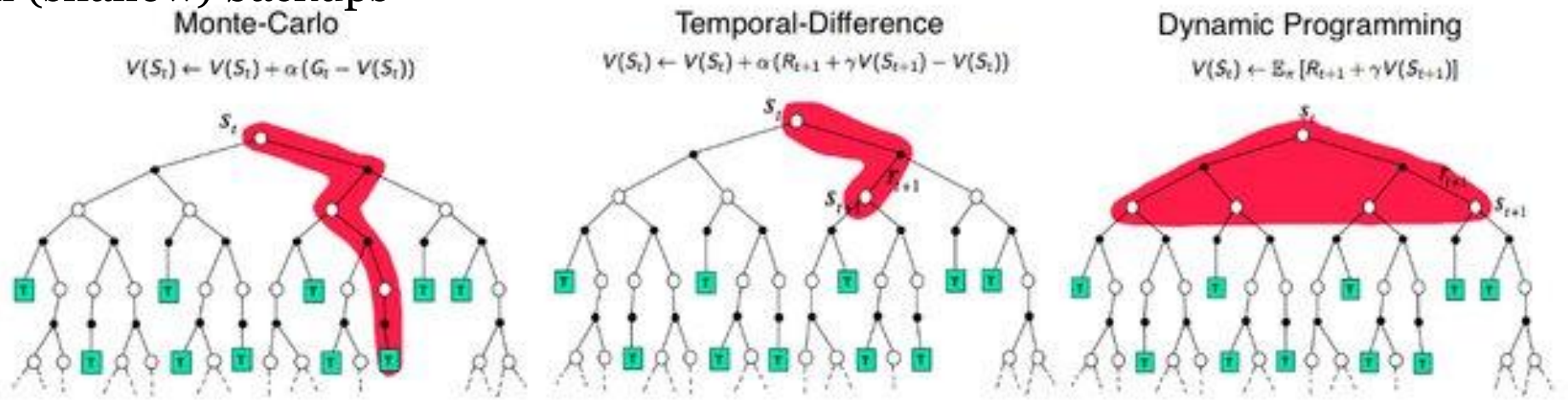
$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1})\big) \qquad \text{for state-value function}$$

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})\big) \qquad \text{for action-value function}$$

Monte-Carlo (MC) vs Temporal Difference (TD) Learning

- Recall the state transition in MDP
- Dynamic Programing (DP): known model
- MC & TD: unknown model, or model free
- MC: Full (deepest) backups
- TD: partial (shallow) backups





Monte-Carlo (MC) vs Temporal Difference (TD) Learning

- Monte-Carlo

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(G_t - Q(S_t, A_t)\big)$$

  - G_t is the cumulated reward throughout the state transition until the end, i.e.,

- Temporal-Difference (SARSA)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big)$$

  - $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ is the expected return with immediate reward and estimated value of the next state
  - The estimated value is not 100% sure, so discounted by γ

- Temporal-Difference (Q-Learning, or SARSAmax)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t)\right)$$

Q-Learning (revisit)

- On-Policy / Off-Policy ?
- Behavior policy / Target policy ?
- Recall Q-learning update rule
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$
- $Q(s_t, a_t) = \int_0^{2\pi} \big(R_t + \gamma Q(s_{t+1}, a_{t+1}) \big) p(a_{t+1}|s_{t+1}) p(s_{t+1}|s_t, a_t) ds_{t+1}, a_{t+1}$
  - Target: (greedy action) $p(a_{t+1}|s_{t+1}) = \delta(a_{t+1} - a^*_{t+1})$ or $a^*_{t+1} = \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$
  - Behavior: (ϵ-greedy for exploration)
- TD-target: $R_t + \gamma \max Q(s_{t+1}, a_{t+1})$
- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$

Q-Learning vs SARSA

	Q-learning	SARSA
Learning Type	Off-Policy	On-Policy
Next action decision	Based on the "best" action in a set of actions a'	Based on policy π (e.g., ϵ-greedy policy)
Q-table update rule	Based on the greedy policy from the Q-table	Based on the current state, current action, obtained reward, next state, and next action
Convergent	Converged to an optimal solution under the assumption that, after generating experience and training, the system switches over to the greedy policy	Converged to an optimal solution under the assumption that the system keeps following the same policy that is used to achieve the experience
Application cases	Preferable in situations where the agent's performance is not considered during the training process, but switches to learn an optimal greedy policy eventually	Preferable in situations where an agent's performance is taken into consideration during the process of learning and generating the experience
Popularity	More popular	Less popular

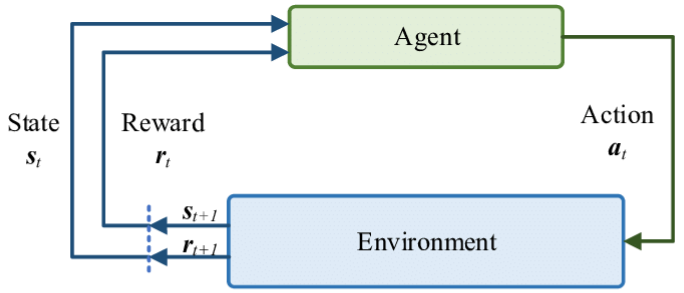
DEEP REINFORCEMENT LEARNING

We've learned so far ..

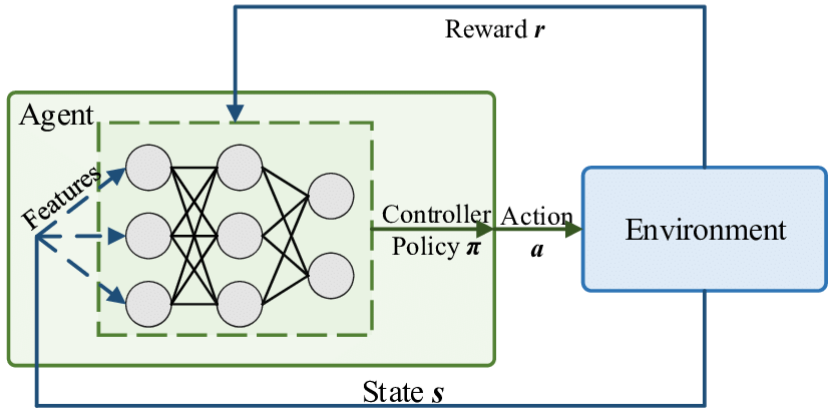
- What is Reinforcement Learning?
- Markov Decision Process
- Fundamentals of RL
- State-Value Function and State-Action Function
- How to train? Monte Carlo and Temporal Difference Methods
- Q-Learning and on / off-policy

Why Deep Reinforcement Learning?

- Limitations of RL
  - Limited capability for high-dimensional state and spaces
  - Requires manual feature engineering
  - Often infeasible for complex environment due to the "curse of dimensionality"
- Deep RL?
  - Combining RL with Deep Learning, usually neural networks
  - NN is used to approximate the value functions, policies, or models of environment



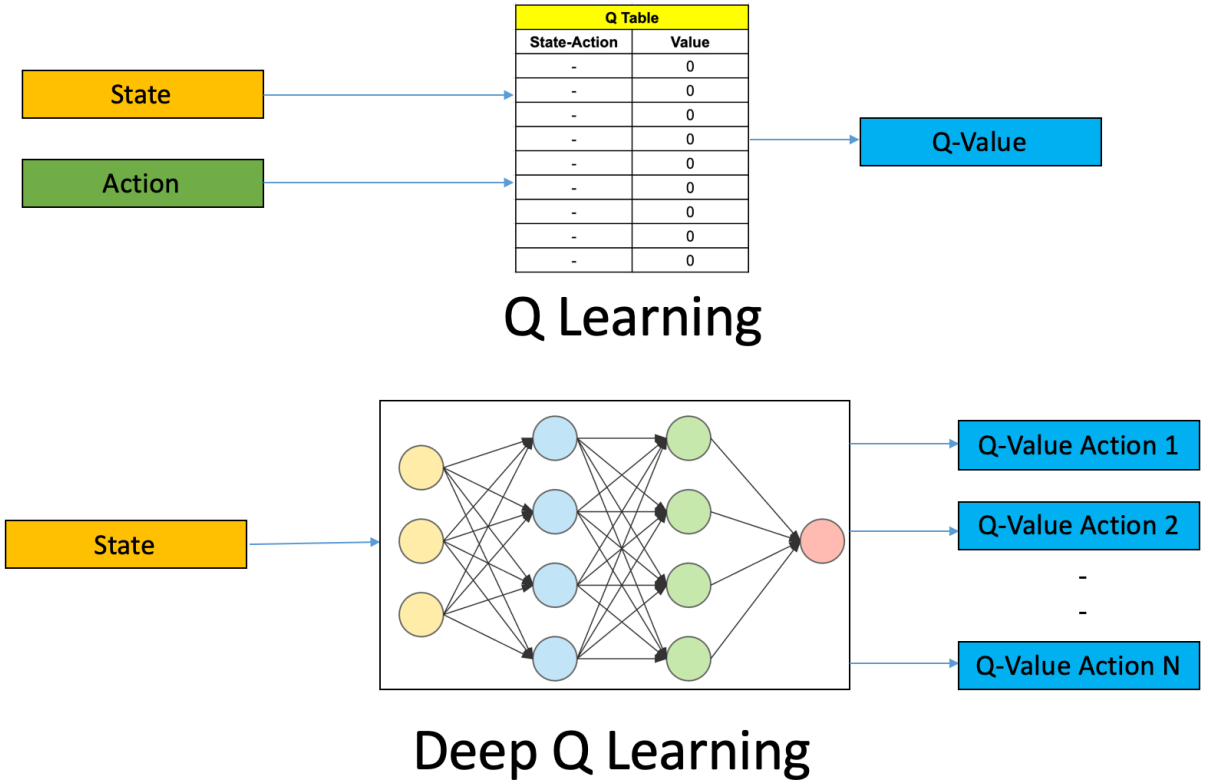
(a)



(b)

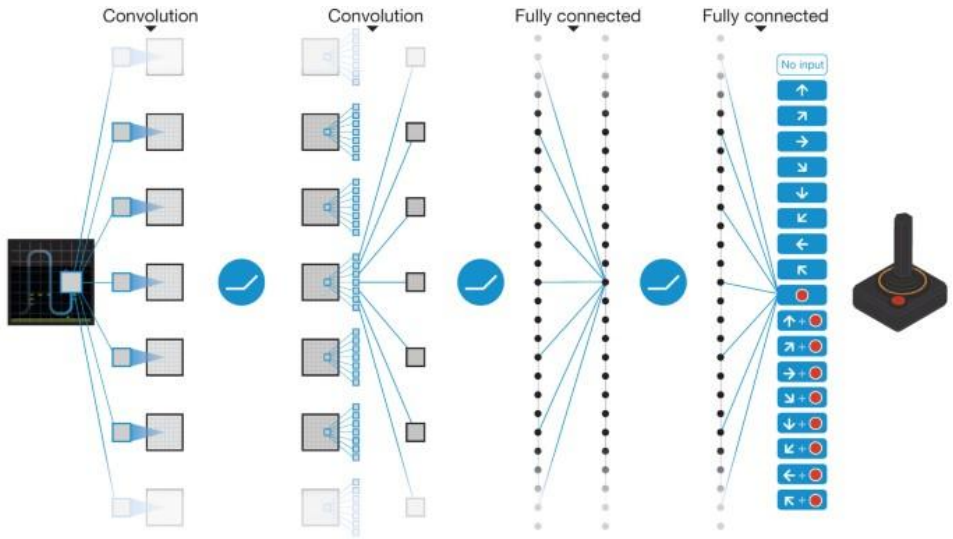
Why Deep Reinforcement Learning?

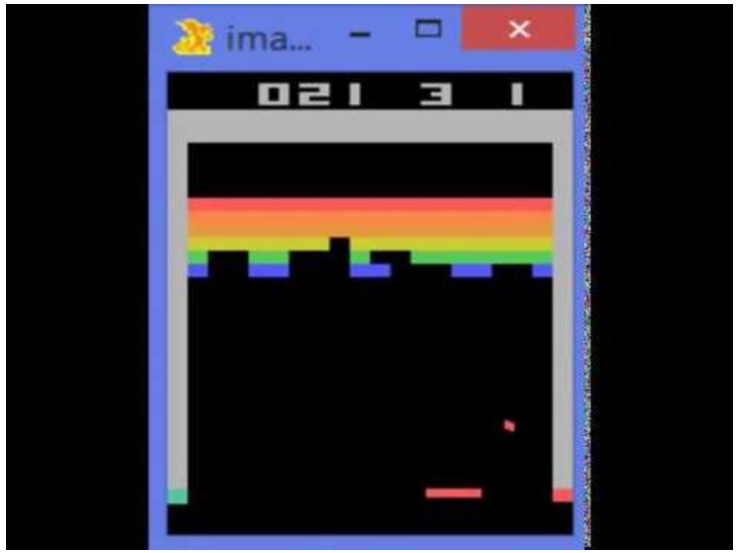
- DRL can be useful for
  - Complex Decision Making
  - High-Dimensional Input Handling
  - Adaptability and Flexibility
  - Human-Level Performance
  - Real-World Application
- Key Components of DRL
  - Neural Networks
  - Experience Replay
  - Target Networks



Why Deep Reinforcement Learning?

- DRL can be useful for
  - Complex Decision Making
  - High-Dimensional Input Handling
  - Adaptability and Flexibility
  - Human-Level Performance
  - Real-World Application
- Key Components of DRL (DQN, Deep Q-Network)
  - Neural Networks
  - Experience Replay
  - Target Networks

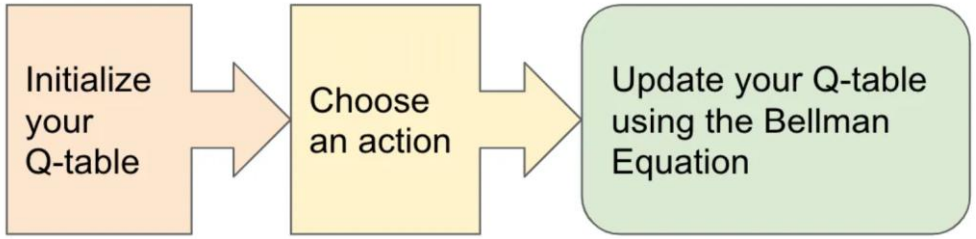




https://www.youtube.com/watch?v=V1eYniJoRnk

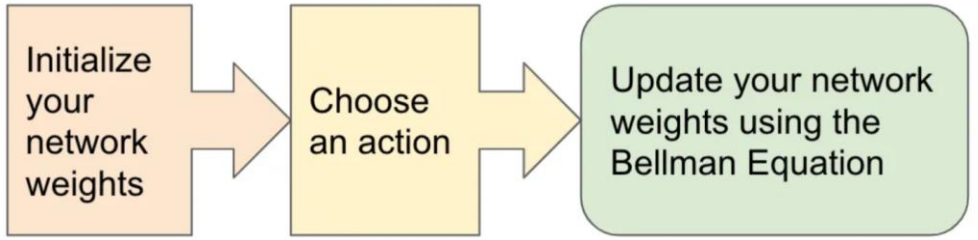
Combining Q-Learning with Neural Networks

- Q-Learning



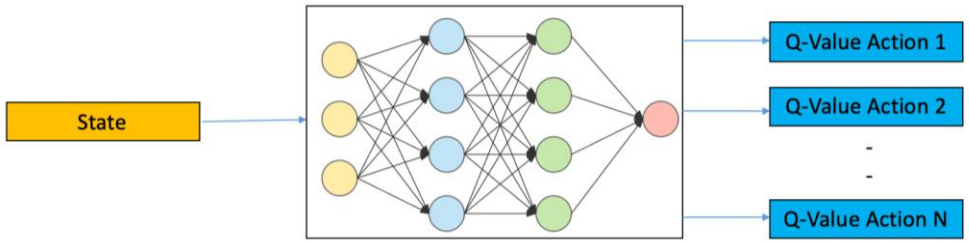
  - Uses Q-table and Bellman Equation
  - Infeasible for large state-action spaces due to memory and computation constraints
  - Requires discrete and relatively small state and actions spaces (e.g., grid world)

- Deep Q-Learning



  - Uses NN to approximate Q-value function (Regression with DNN)
  - Remarks
    - Deep Q-Network (DQN)
    - Experience Replay
    - Target Network
  - Suitable for large, continuous state-action spaces

Deep Q-Network



- Loss Function:
  minimize the difference between the predicted Q-values and the target Q-values

$$L_\theta = \left(y - Q(s_t, a_t, ; \theta)\right)^2$$

$$y = r_{t+1} + \gamma \max_{a'} Q_\theta(s_{t+1}, a'; \theta^-)$$

where θ^- is the parameters for tarket network

- Target is to minimize the loss function : $\theta \leftarrow \theta - \alpha \nabla_\theta L_\theta$
- Deep Q-Learning Steps
  1. All the past experience is stored in memory
  2. The next action is determined by the maximum output of the Q-network
  3. The loss function is MSE of the predicted Q-value and the target Q-value

DQN 2013 (Playing Atari with Deep Reinforcement Learning)

- Raw pixel image (84 x 84 x 4, reduced size, 4 consecutive frames)
- Ping pong, Breakout, Galaxy, …
- Contributions
  - Experience replay
    - off policy
    - Mini-batch SGD
    - State correlation is reduced
  - CNN uses → like human

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

https://www.youtube.com/watch?v=V1eYniJ0Rnk

DQN 2015 (Human-level control through deep reinforcement learning)

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function Q with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

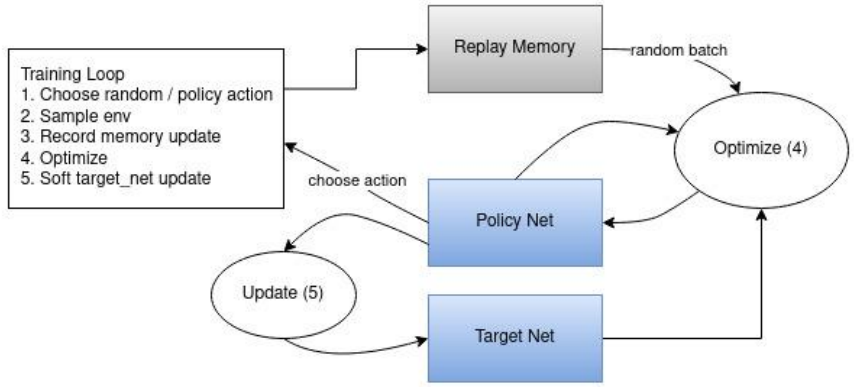
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

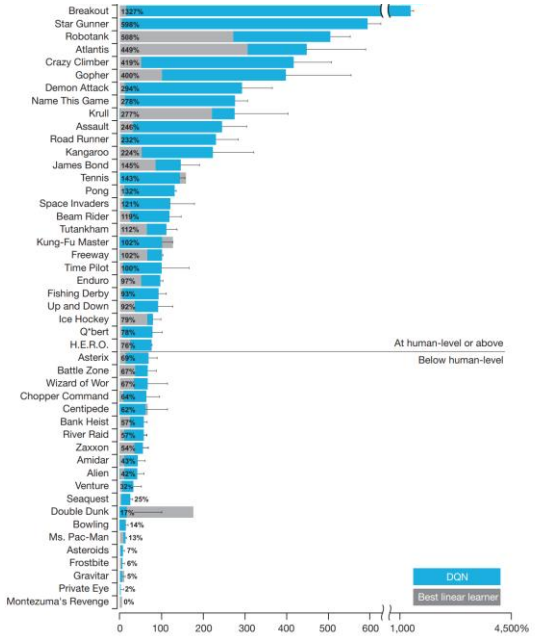
    Every C steps reset $\hat{Q} = Q$

  End For

End For

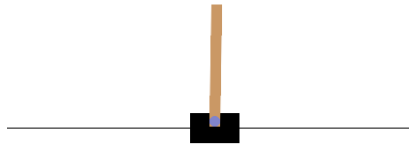
- Mostly the same as DQN 2013
- Target network / main network
- Sparse update of target network





Implementation Example: Cart-pole

- Variables
  - x : Position of the cart along the track
  - $\dot{x}$: Velocity of the cart
  - θ : Angle of the pole from the vertical (upright) position
  - $\dot{\theta}$: Angular velocity of the pole
  - F : Force applied to the cart
  - g: Acceleration due to gravity (9.8 m/s²)
  - m_c : Mass of the cart
  - m_p : Mass of the pole
  - l: Half-length of the pole (distance from pivot to the pole's center of mass)
  - I : Moment of inertia of the pole around its center of mass



- Governing Equations (mechanics)

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2 \sin\theta = F$$

$$(I + m_p l^2)\ddot{\theta} + m_p l\ddot{x}\cos\theta = m_p g l \sin\theta$$

- Simplified Equations (small angle assumption)

$$\ddot{x} = \frac{F + m_p l(\ddot{\theta} - \dot{\theta}^2\theta)}{m_c + m_p}$$

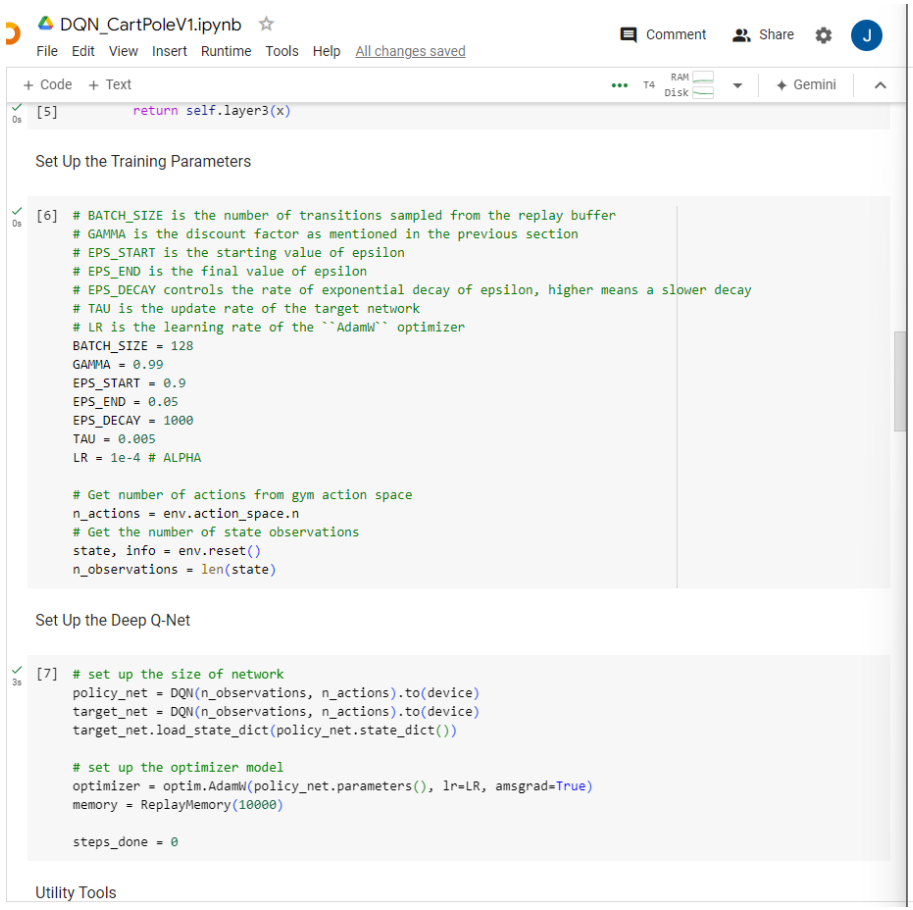
$$\ddot{\theta} = \frac{m_p l(\ddot{x} - g\theta)}{I + m_p l^2}$$

- Not easy to set up the problem.. Then double pendulum?

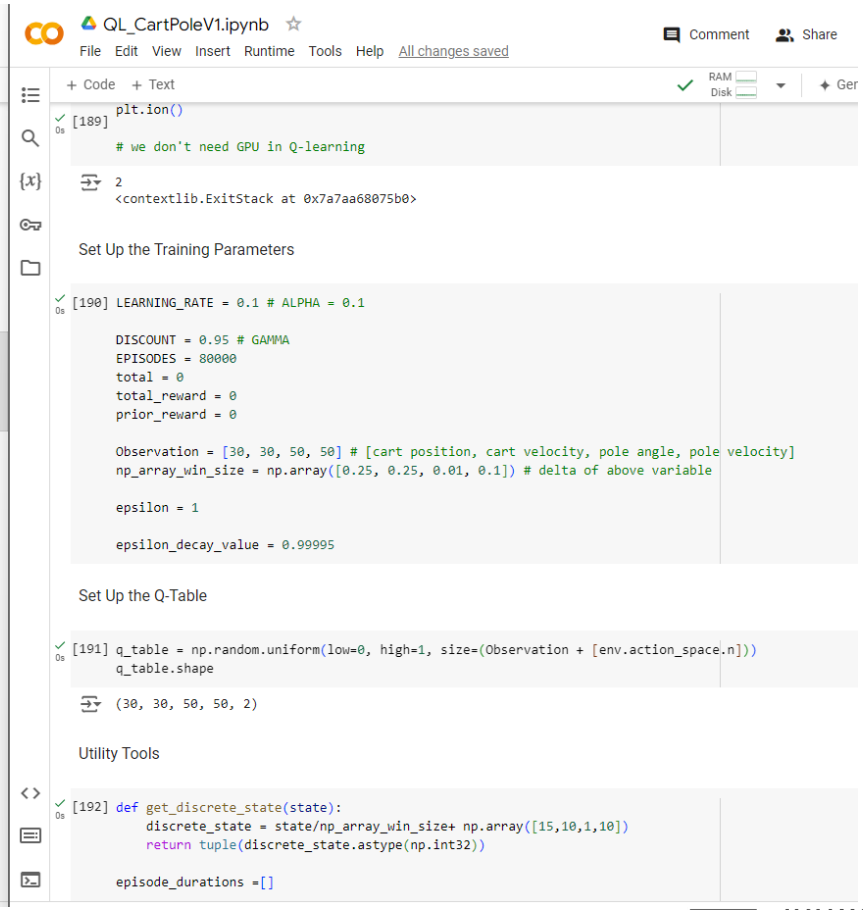
  (inverted pendulum: https://en.wikipedia.org/wiki/Inverted_pendulum)

Implementation Example: Cart-pole (Check out the code)

- DQN

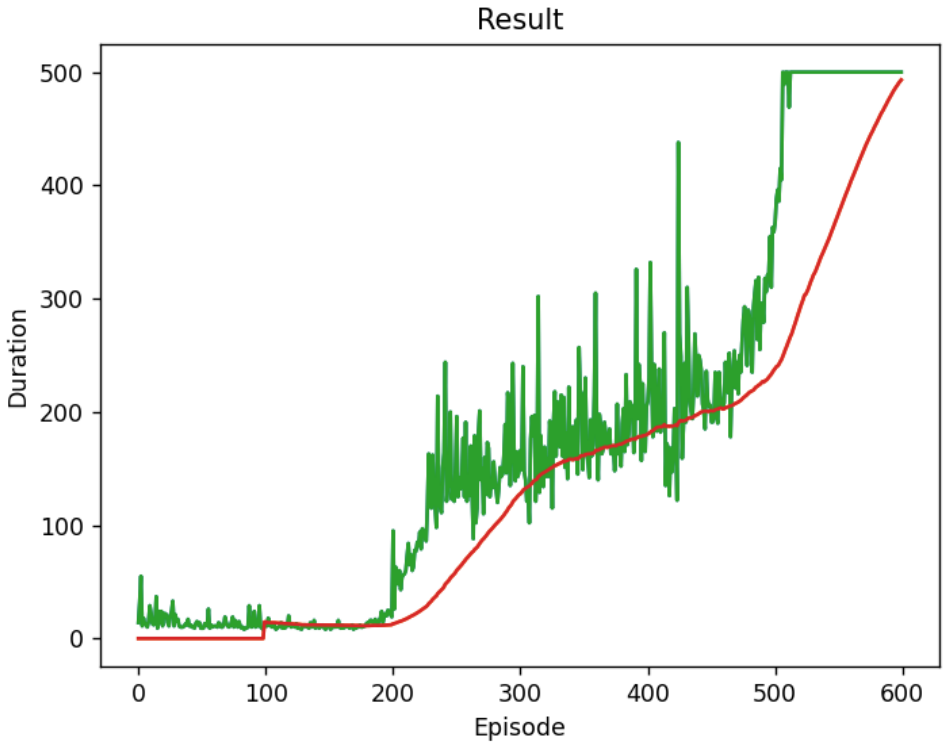


- Q-Learning

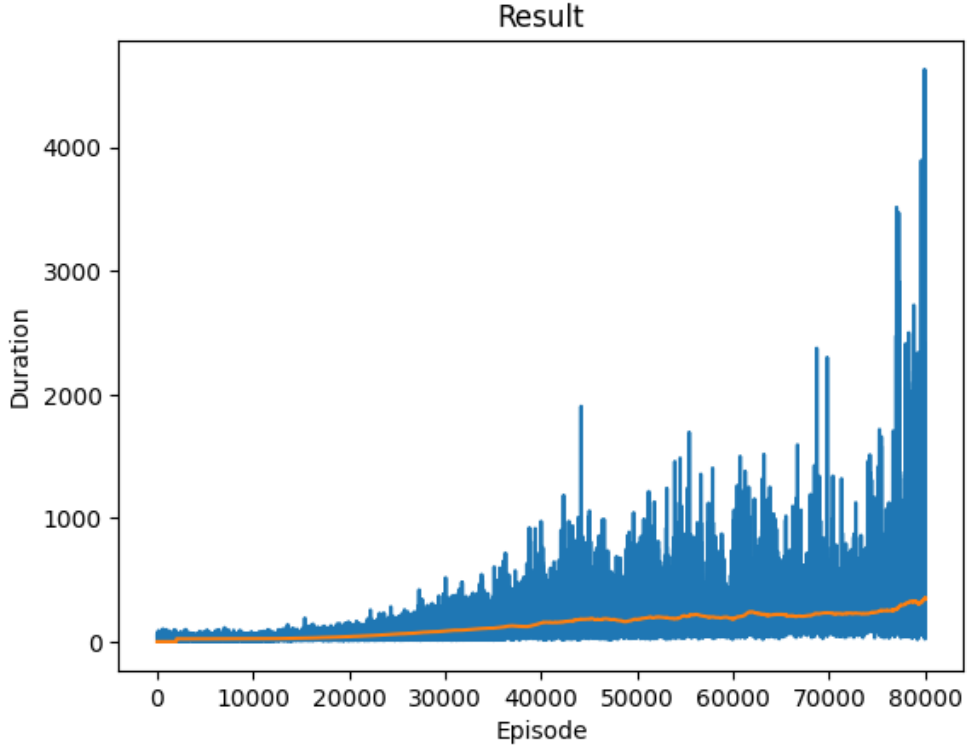


Implementation Example: Cart-pole

- DQN

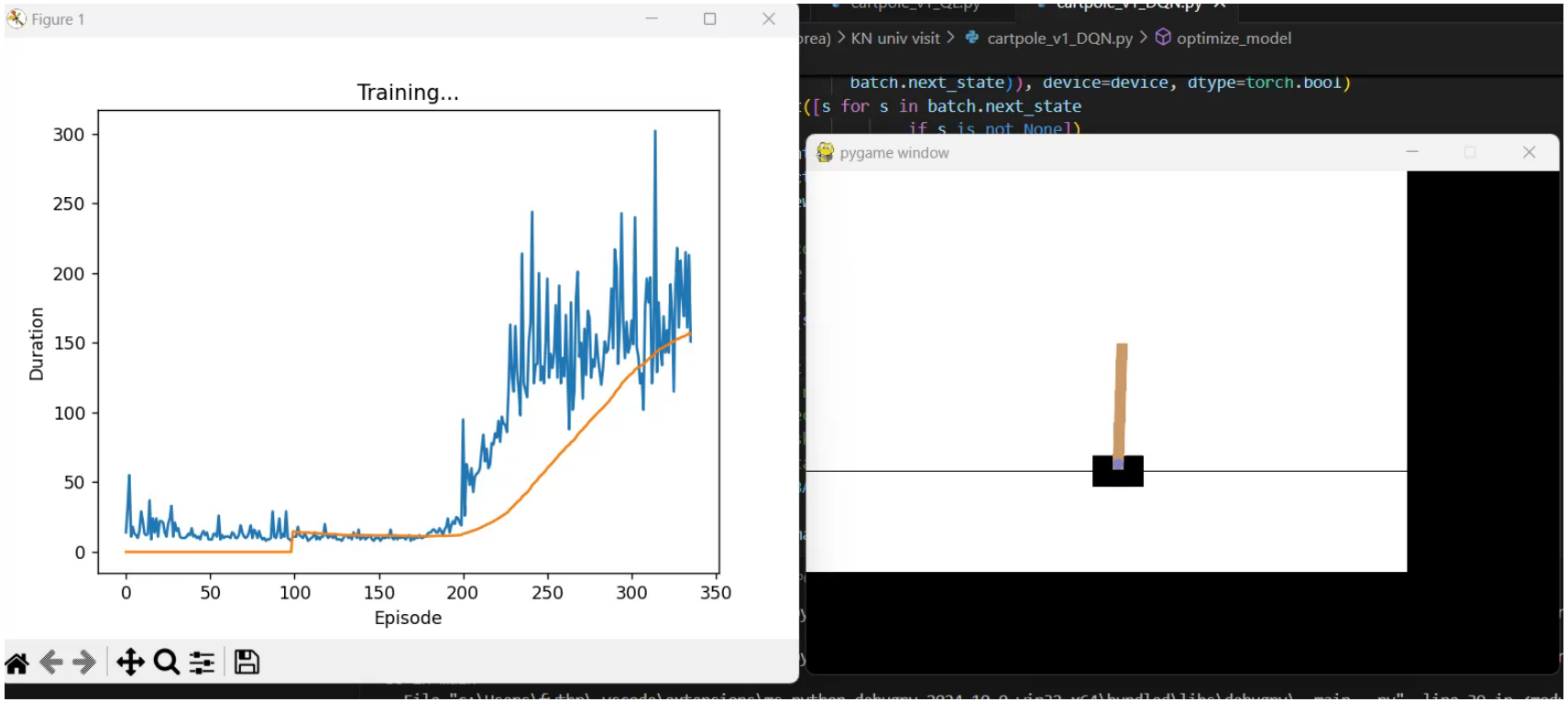


- Q-Learning



Implementation Example: Cart-pole

- DQN
- Q-Learning



Double DQN (Deep Reinforcement Learning with Double Q-learning)

- In DQN, max Q may overestimate the values (too optimistic)
- Theorem 1: "$\max Q(s,a) - V_*(s)$" has always positive error with lower bound of

$$\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}}$$

- DQN: greedy action is from target network

$$R_t + \gamma \max Q_{\theta^-}(s_{t+1}, a_{t+1})$$

- DDQN: greedy action is from main network

$$R_t + \gamma Q_{\theta^-}\left(s_t, \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1})\right)$$

Chapter 5. Introduction To Robot Operating System (ROS)

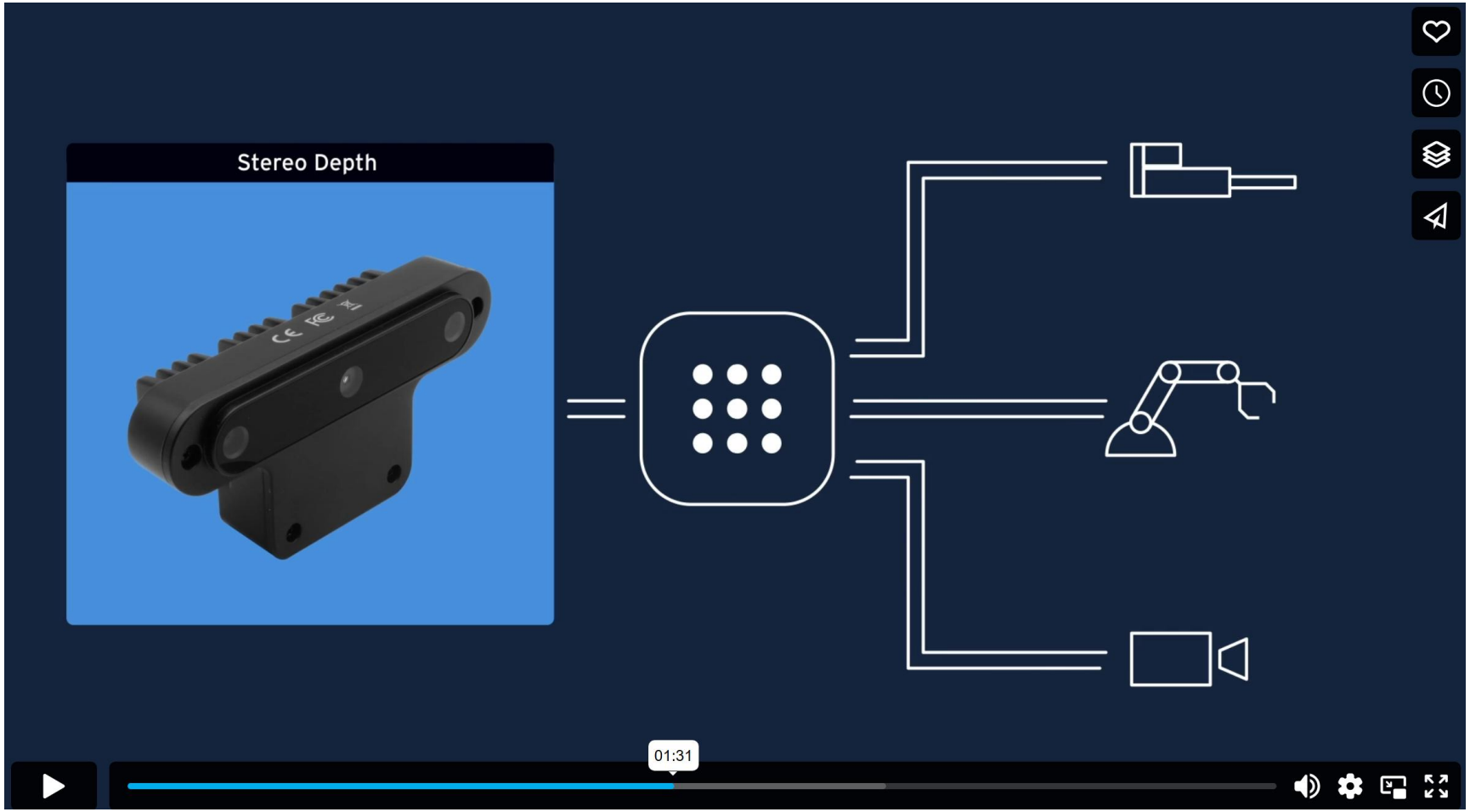
Jeong-woo Han, PhD
Mechanical and Mechatronics Engineering

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Course Overview

- In this lecture, you will learn
  - Overview introduction to ROS (Robot Operation System)
    - What is ROS and Why ROS?
    - Fundamentals of ROS
  - Setting Up a Simulation Environment with a Virtual Machine on Windows
  - An exemplary simulation on mobile robot navigation

What is Robot Operation System (ROS)



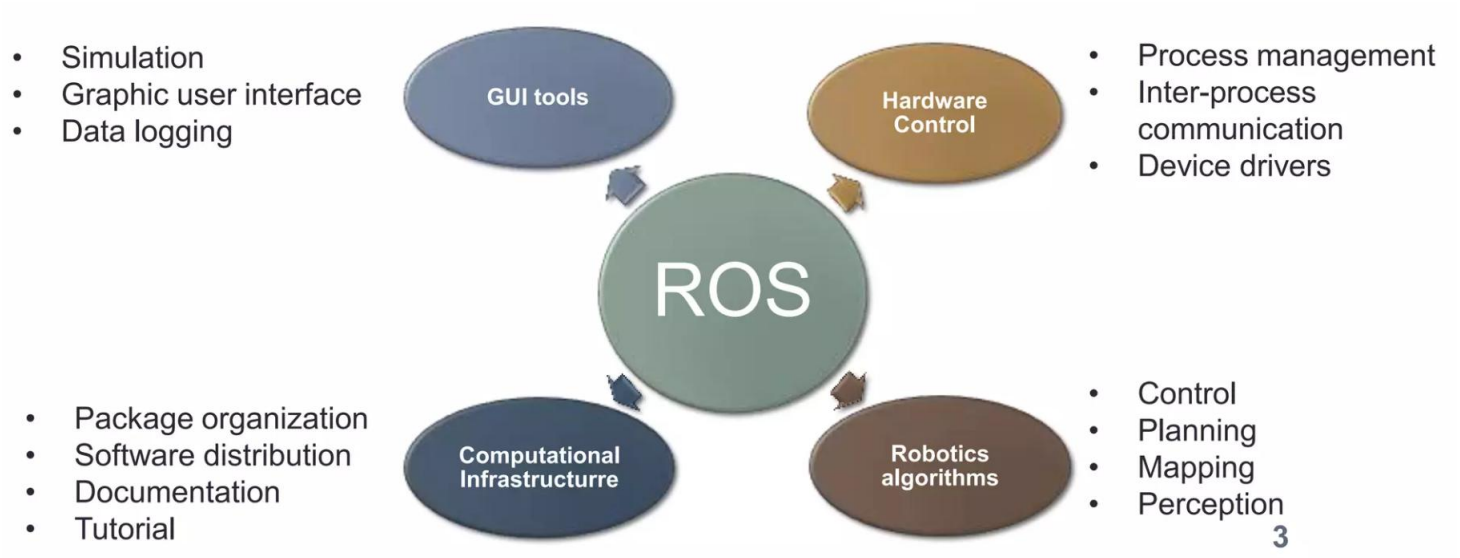
https://vimeo.com/639236696

What is Robot Operation System (ROS)

- Open-source robot operating system
- A common SW platform with a set of SW libraries and tools
- For robot applications that work across a wide variety of robotic platform
- Developed in 2007 at Stanford
- Since 2013, managed by OSRF (Open Source Robotics Foundation)

ROS

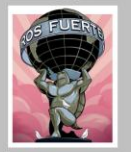
Open Source Robotics Foundation



What is Robot Operation System (ROS)

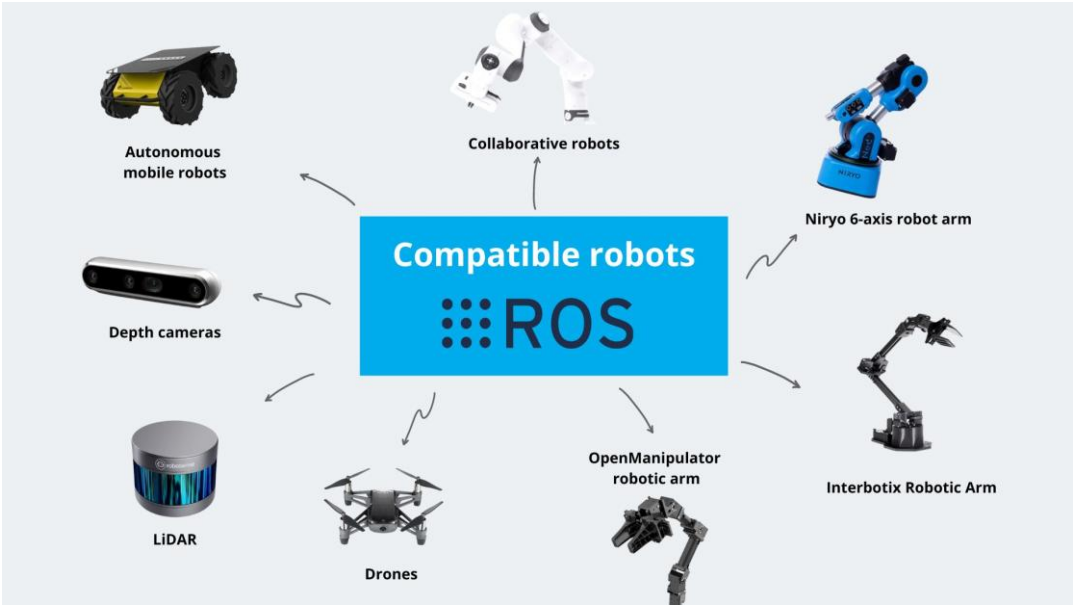
- ROS release is supported on one Ubuntu LTS
  - Kinetic with Ubuntu 16.04 LTS
  - Noetic with Ubuntu 22.04 LTS

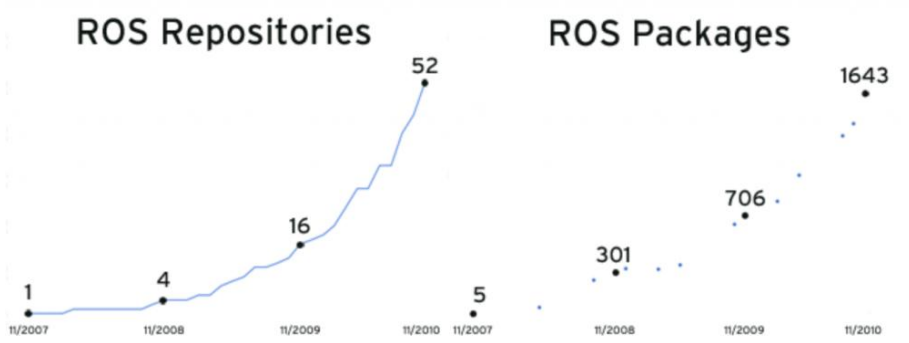
Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			June 27, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)

ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010	 Box Turtle		--

Why ROS?

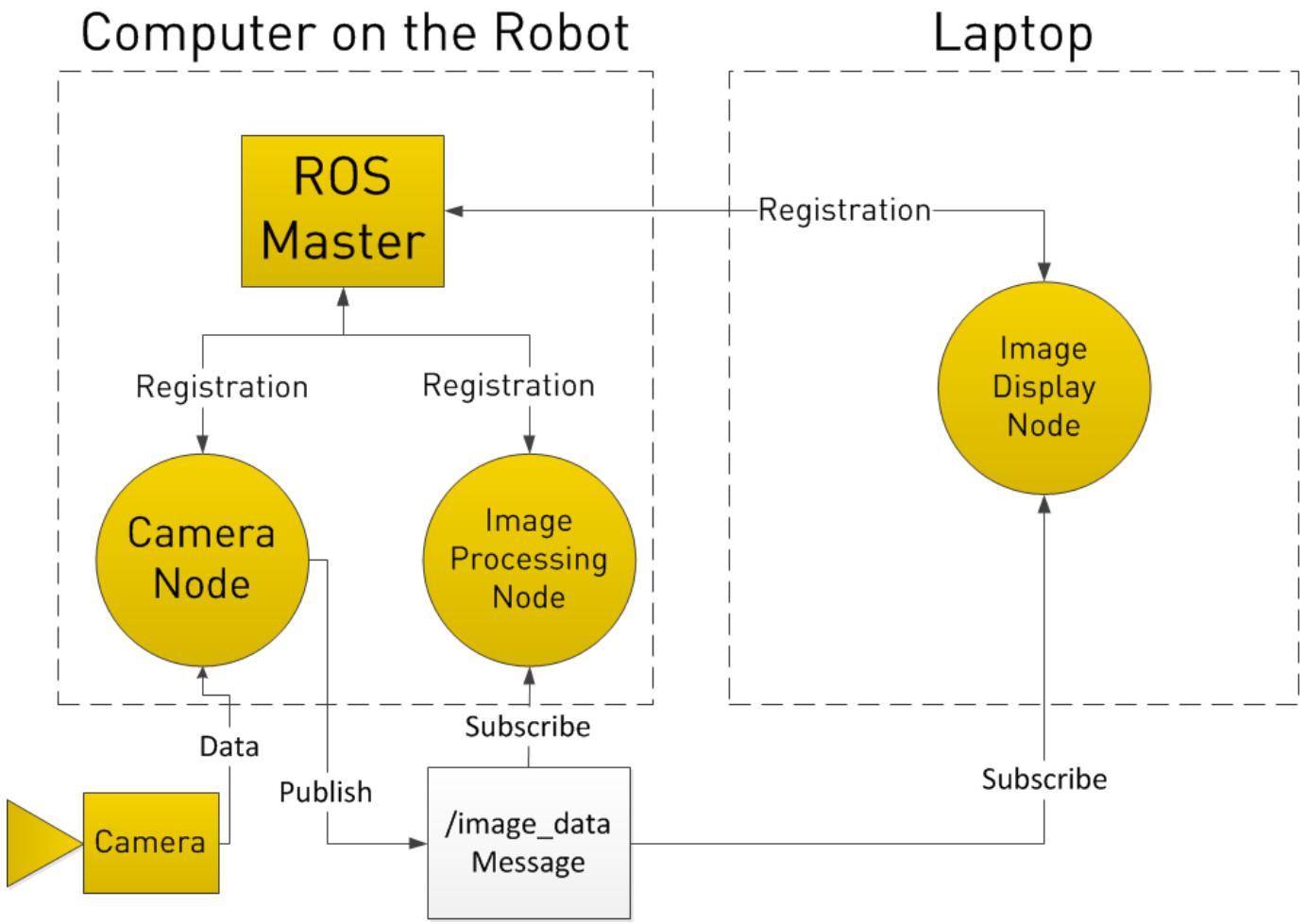
- **Simplifies Low-Level Coding**:
  Writing low-level code for robotic applications is cumbersome.
- **Component Reusability**:
  Facilitates the reuse of components.
- **Open Source**:
  Leverages the power of the open-source community.
- **OS Abstraction**:
  Your app interfaces with the OS, not directly with the hardware.
- **Modular Architecture**:
  A small core package, with the rest as distributed open packages.
- **Collaborative Framework**:
  Enables groups to work on complex projects with a common, well-organized framework, adding minimal overhead.
- **Rich Bash Command Set**:
  Includes powerful bash commands like roscd, rospack, rosmake, rosnode, rostopic.
- **Language Independence**:
  Supports multiple programming languages (RosCPP, RosPY).
- **Easy Training**





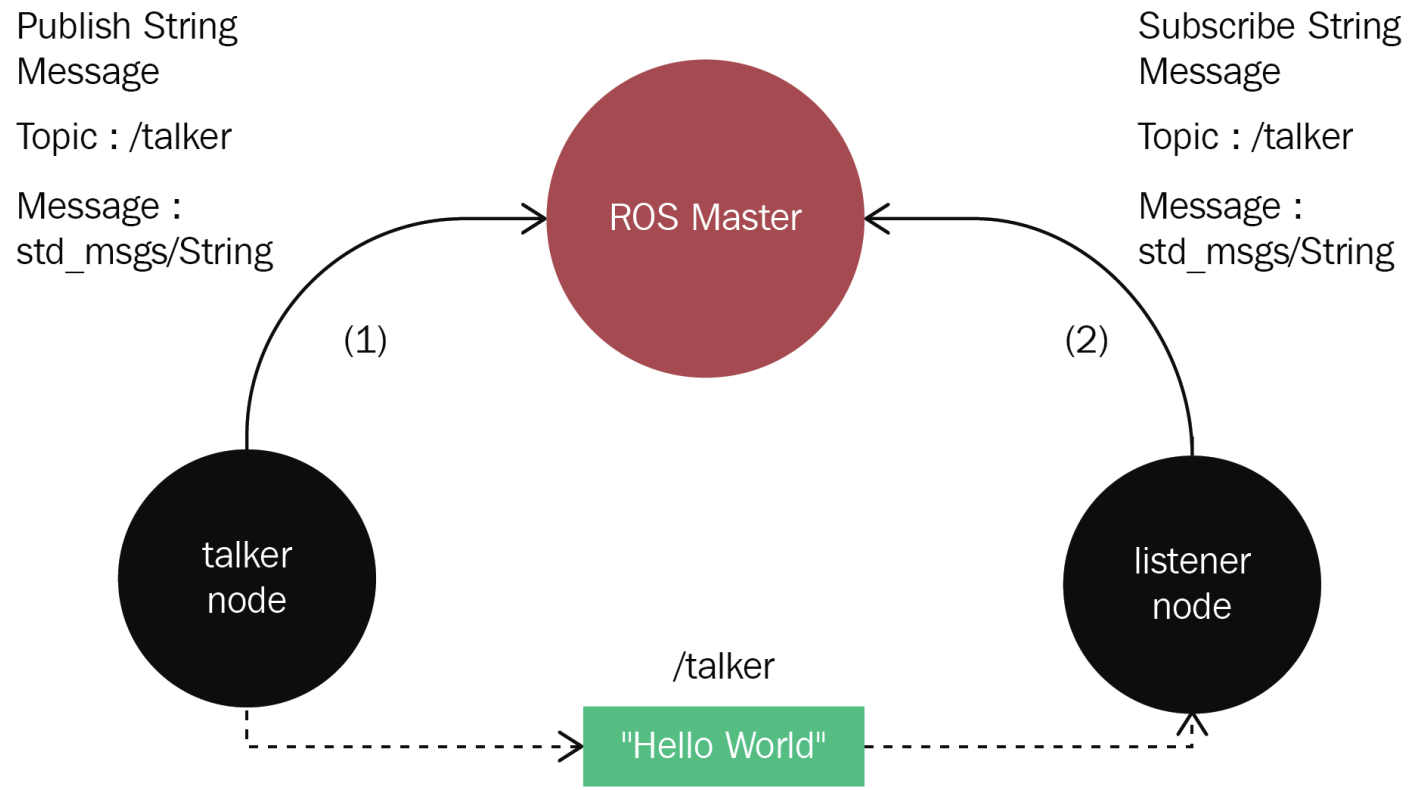
ROS Fundamentals: Architecture

- Master
- Nodes
- Topics
- Messages



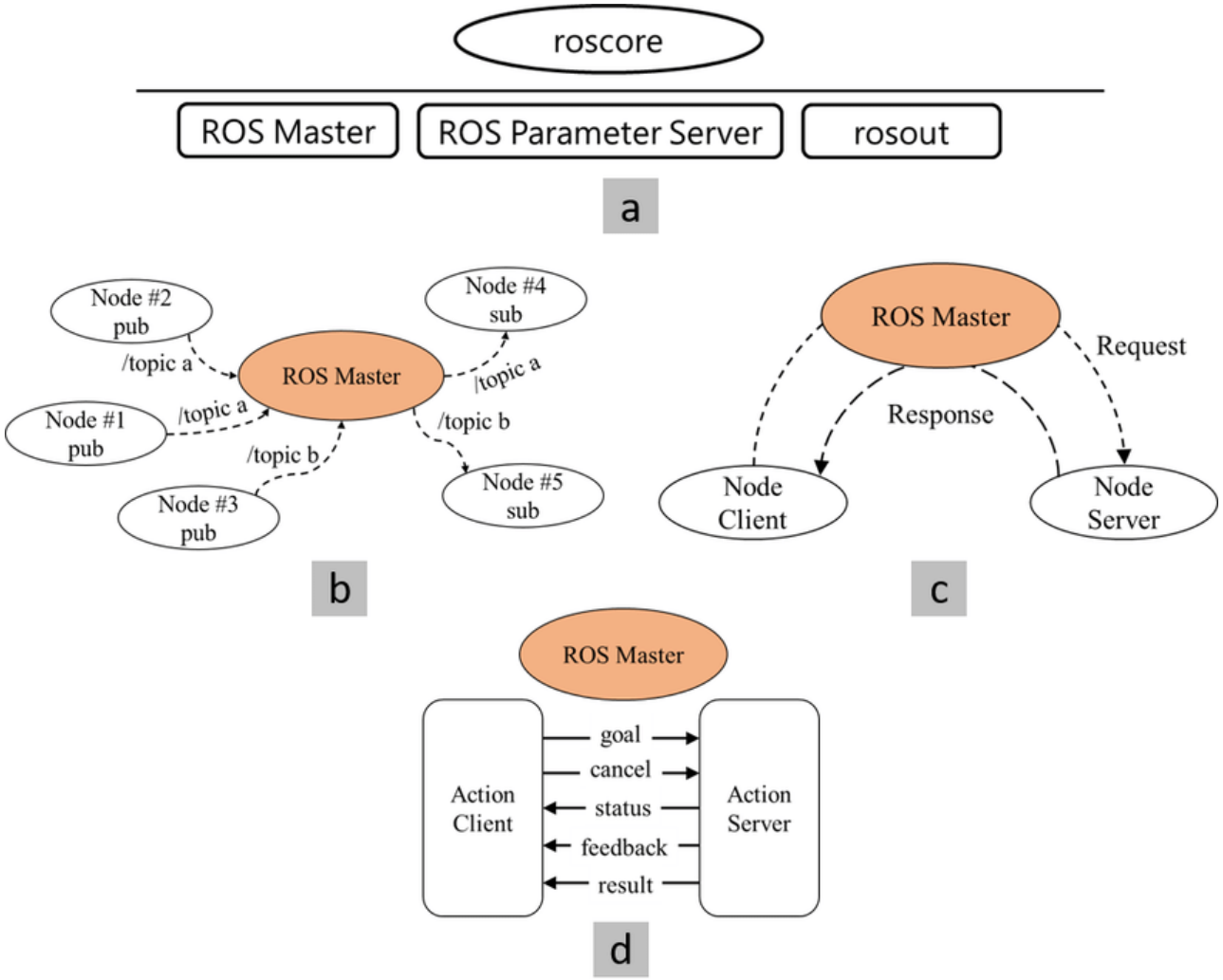
ROS Fundamentals: Architecture

- Master
- Nodes
- Topics
- Messages



ROS Fundamentals: Communication

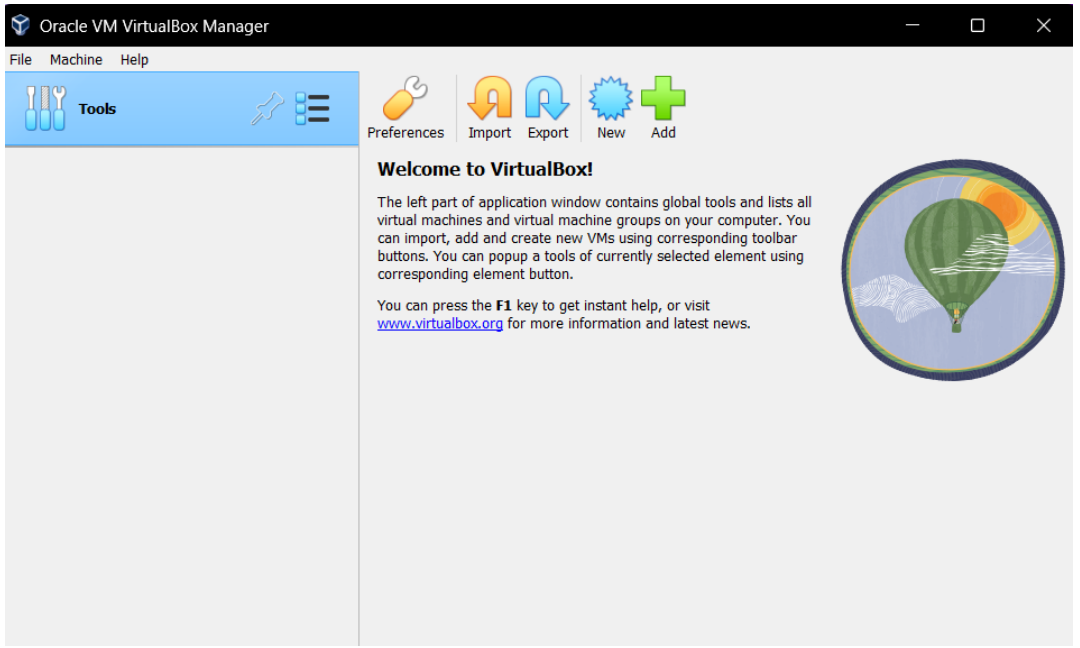
- Nodes
  - Publisher (talker)
  - Subscriber (listener)
- Topic
- Service
- Action



Setting Up ROS on a Virtual Machine

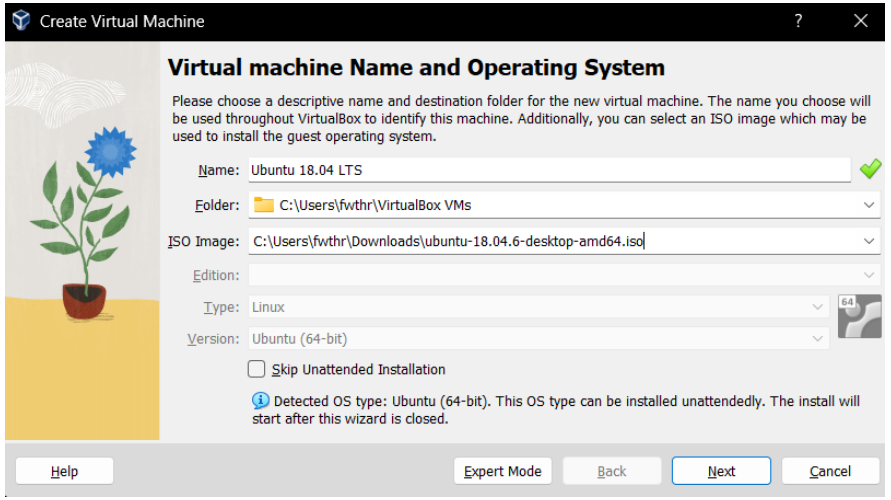
- Virtual Machine (VM) on Windows
  - Go to https://www.virtualbox.org/

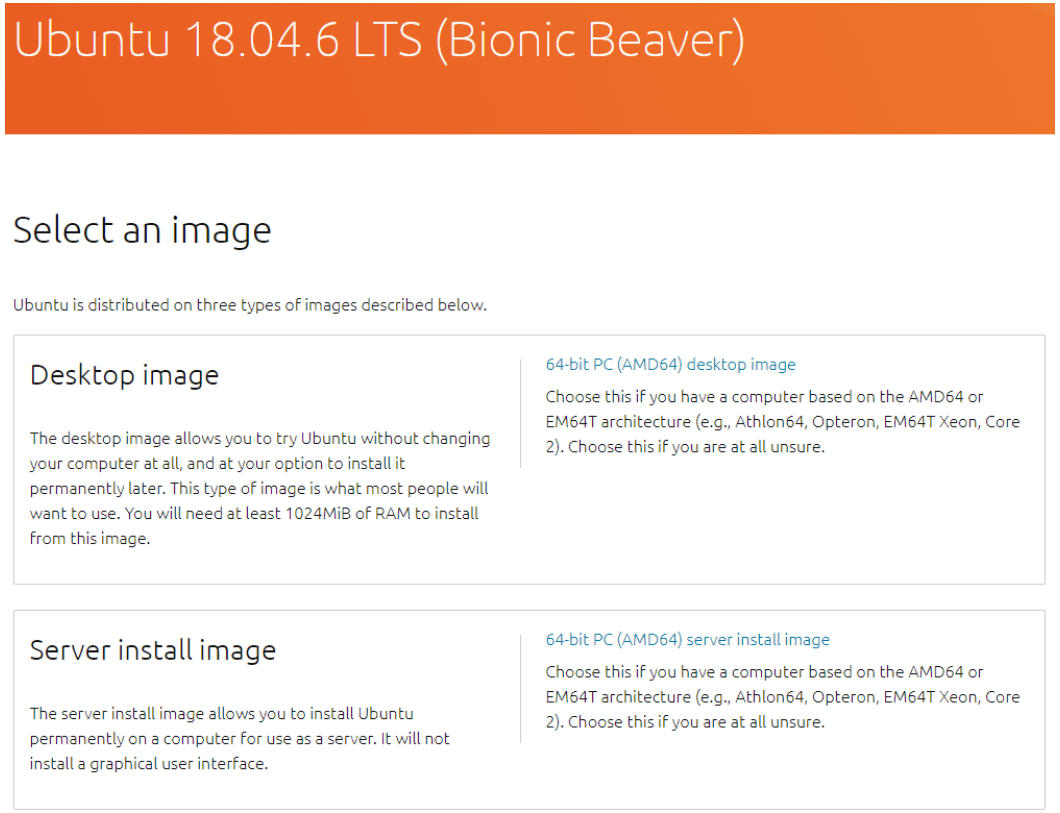
  - Click "*Downloads*" on the left tab, select "*Windows hosts*"
  - Install *VM* with instructions



Setting Up ROS on a Virtual Machine

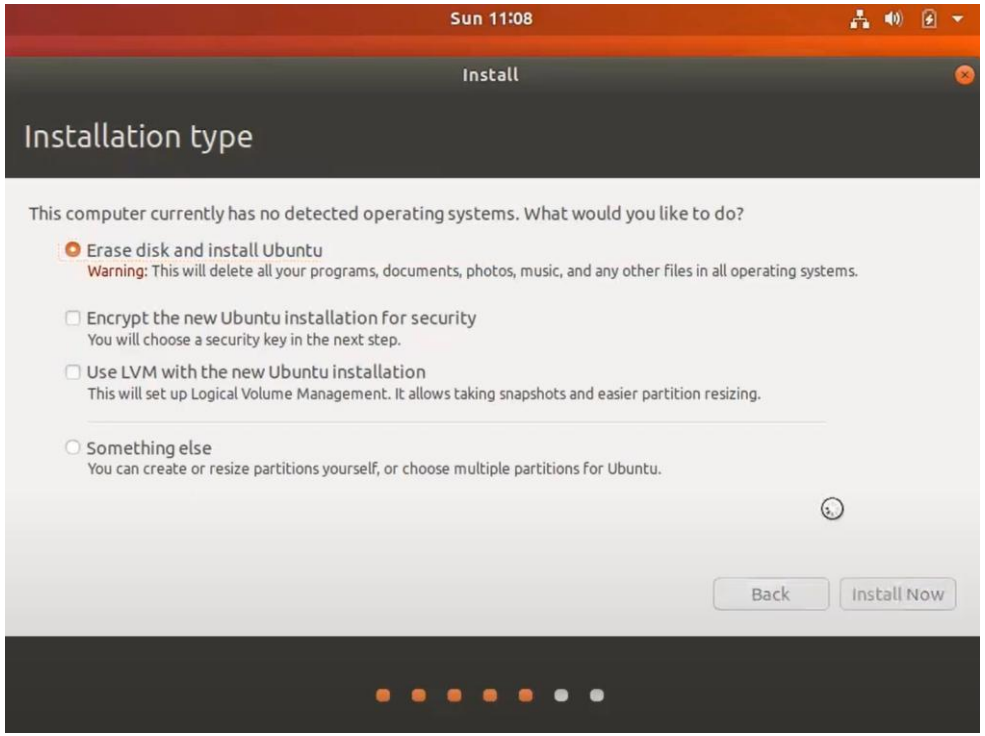
- Install Ubuntu 18.04 LTS on VM
  - Go to https://releases.ubuntu.com/18.04/
  - Download desktop image on your laptop
  - Click *New* on *VM* to create a new volume
  - Allocate the RAM and Storage

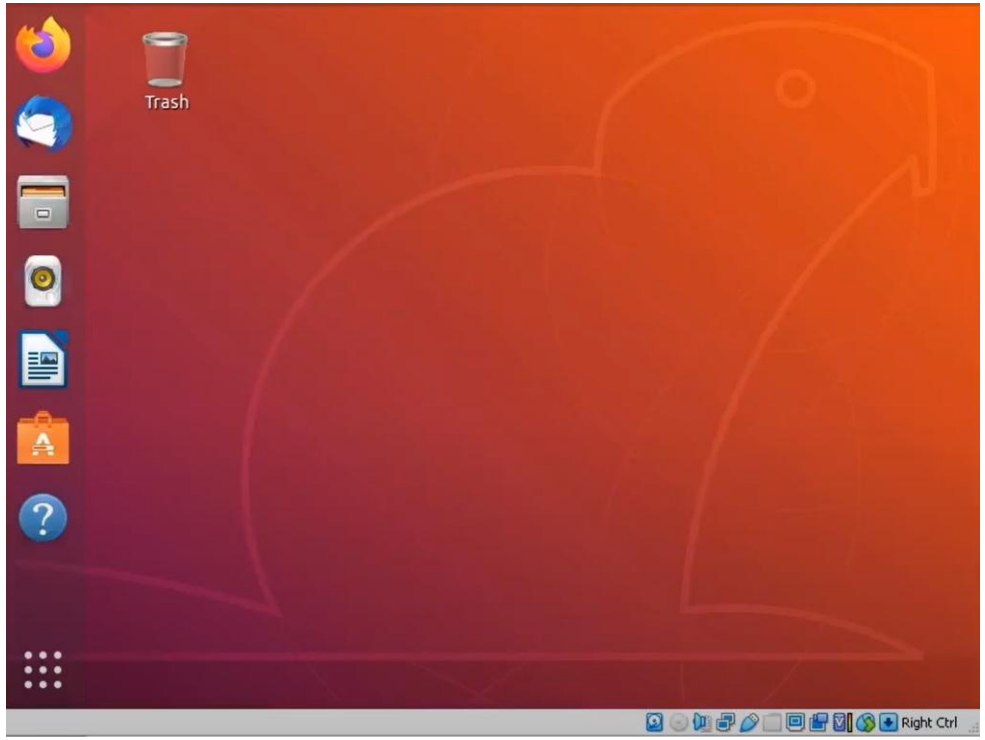




Setting Up ROS on a Virtual Machine

- Install Ubuntu 18.04 LTS on VM





Setting Up ROS on a Virtual Machine

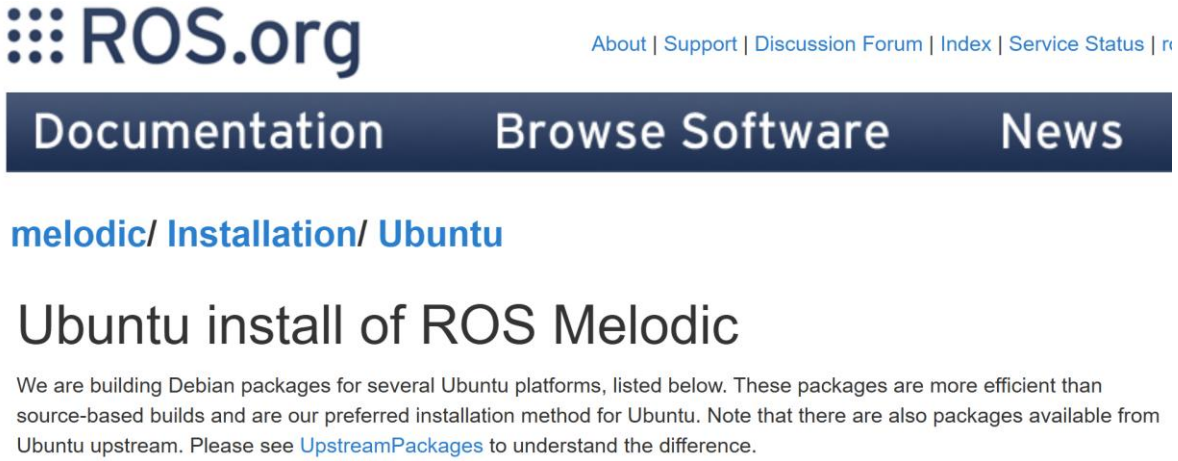
- Install ROS melodic on Ubuntu 18.04 LTS:
  - sudo apt-get update

```
ubuntu-bionic@ubuntubionic-VirtualBox:~$ sudo apt-get update
```

  - sudo apt-get upgrade

```
ubuntu-bionic@ubuntubionic-VirtualBox:~$ sudo apt-get upgrade
```

  - Go to ROS melodic site (melodic/Installation/Ubuntu - ROS Wiki)



Setting Up ROS on a Virtual Machine

- Install ROS melodic on Ubuntu 18.04 LTS:
  - Set up your sources list (1.2)
  - Set up your keys (1.3)
  - sudo apt update
  - Install desktop-full
    sudo apt install ros-melodic-desktop-full

1.2 Setup your sources.list

Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Mirrors Source Debs are also available

1.3 Set up your keys

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

1.4 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt update
```

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

In case of problems with the next step, you can use following repositories instead of the ones mentioned above ros-shadow-fixed

Desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D perception

```
sudo apt install ros-melodic-desktop-full
```

Setting Up ROS on a Virtual Machine

- Install ROS melodic on Ubuntu 18.04 LTS (Cint'd)
  - Environment Setup (1.5)
  - Dependency Packages Installation (1.6)
  - Now you are ready to use ROS

1.5 Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

If you have more than one ROS distribution installed, `~/.bashrc` must only source the `setup.bash` for the version you are currently using.

If you just want to change the environment of your current shell, instead of the above you can type:

```
source /opt/ros/melodic/setup.bash
```

If you use zsh instead of bash you need to run the following commands to set up your shell:

```
echo "source /opt/ros/melodic/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

1.6 Dependencies for building packages

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, rosinstall is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build
-essential
```

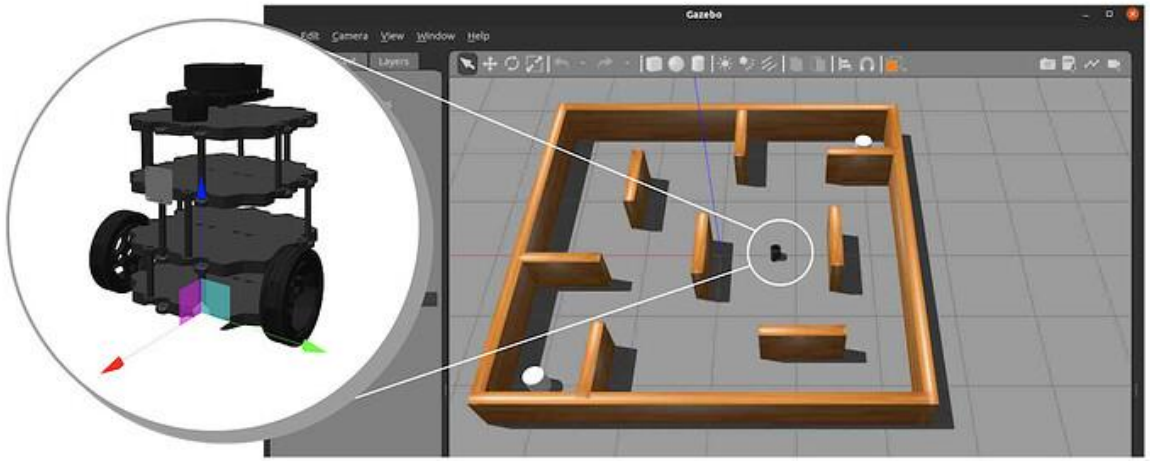
ROS Navigation Simulation (Summit XL)

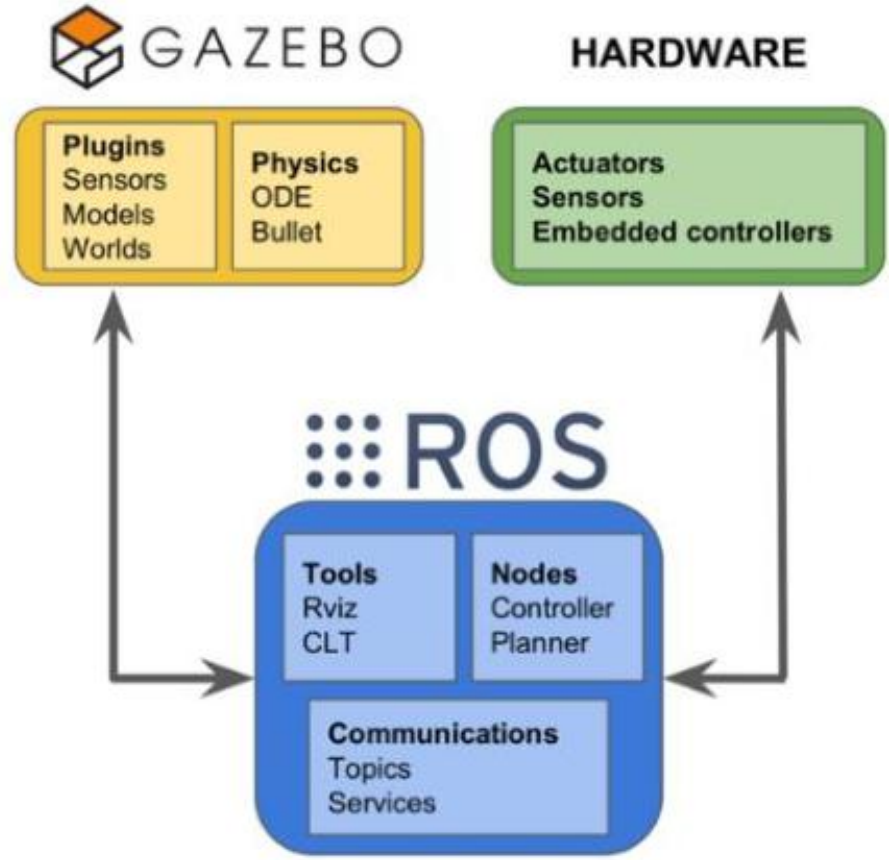
- Install ROS Simulation Packages for Summit XL
  - https://github.com/RobotnikAutomation/summit_xl_sim
  - summit_xl_gazebo (For simulating Env)
  - summit_xl_sim_bringup (For simulation pack)

1. Install dependencies
2. Create a workspace and clone the repository
3. Compile
4. Launch Summit XL simulation with Summit XL option

ROS Navigation Simulation (Summit XL)

- Gazebo (https://gazebosim.org/home)

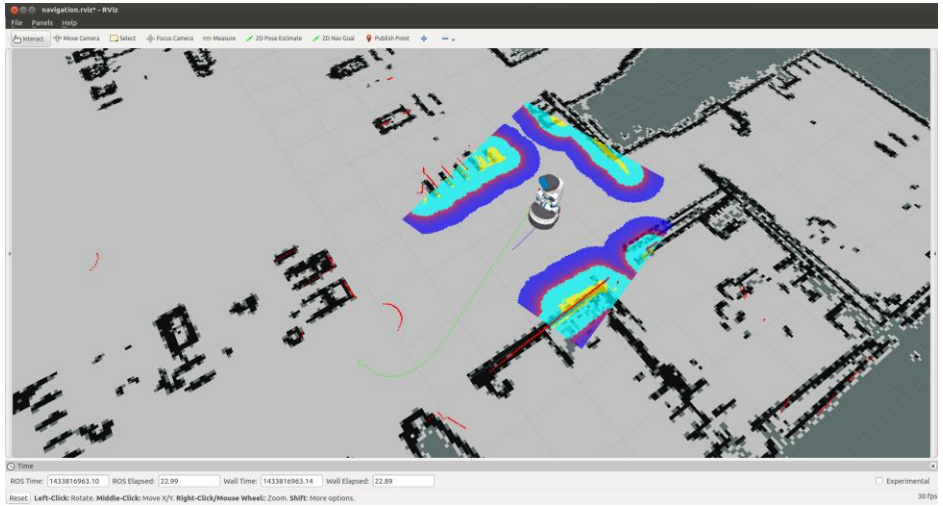


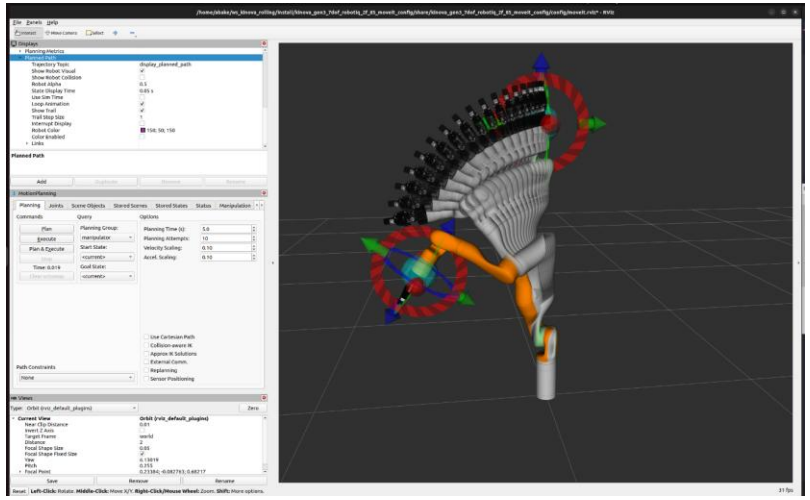


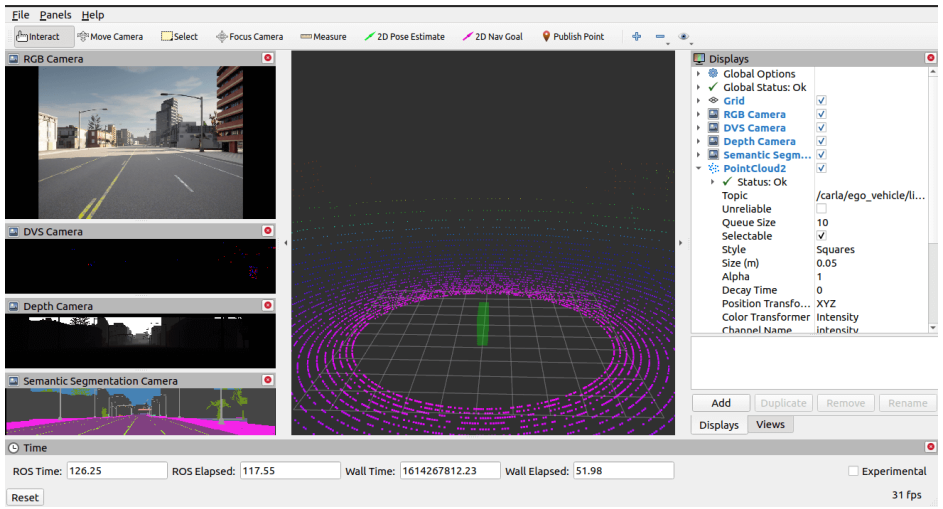
- Physics Env. Simulator commonly used in ROS
- You can simulate control inputs, contact, sensor (with noise), vision, etc like you deploy your robot in a real world

ROS Navigation Simulation (Summit XL)

- Rviz (https://wiki.ros.org/rviz)
  - A visualization tool in ROS
  - You can display your robot, its position, sensors like 2D and 3D lidar, image, depth, ultrasound, etc.







ROS Navigation Simulation (Summit XL)