10-403: Recitation 1

Spring 2024

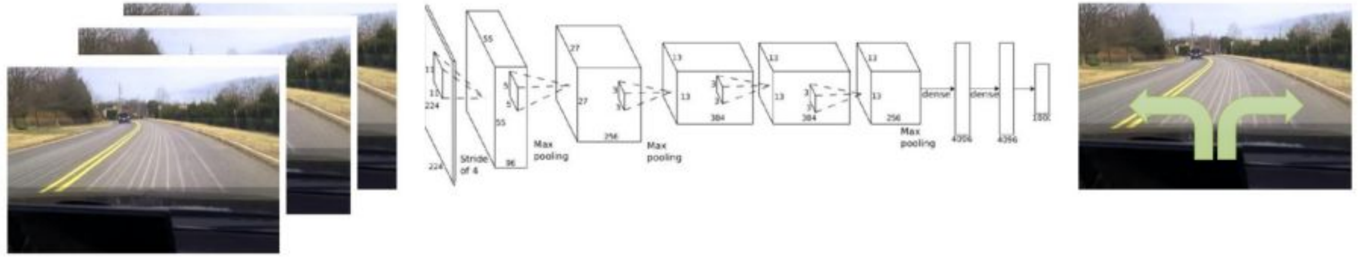
Anish & Alex

Topics We Will Be Covering

- Neural Networks
    - MLPs, CNNs, RNNs, Transformers
- Types of Learning
- Pytorch
- Tensorflow
- Gymnasium

Why deep learning?

“Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.” *– LeCun, Bengio, Hinton*



Scales well with large amounts of data
Less reliant on human feature engineering

Source: 10-403 recitation slides, https://cmudeeprl.github.io/403website_s22/assets/lectures/s22/recitation_1-neural_nets.pdf

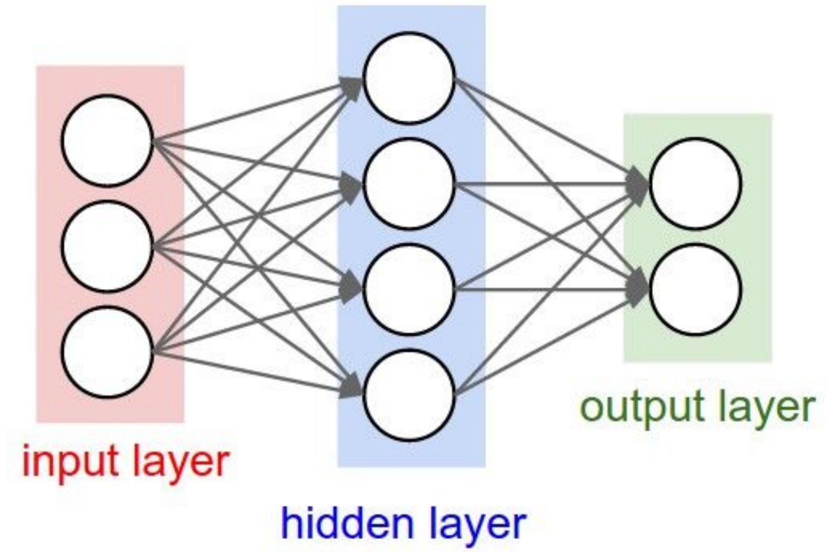
Why “deep” RL?

- Distinction between *classical* and *deep* RL
    - Traditionally, restricted to simpler function approximators (e.g. discrete lookup tables)
        - **Restricted to tabular methods**: problems in which the state and actions spaces are small enough for approximate value functions to be represented as arrays and tables
    - Modern deep learning methods can handle much more complex functions
- Function approximation is super useful for RL
    - Represent agents, reward functions, dynamics models, etc. using deep networks
    - **Handle richer input modalities** like images and text; classical methods often rely on hand-crafted low-dimensional features
- The use of deep learning is most useful in problems with high-dimensional state space
    - This means that with deep learning, Reinforcement Learning is able to solve more complicated tasks with lower prior knowledge because of its ability to learn different levels of abstractions from data

Multi-layer perceptrons (MLPs)

- A universal function approximator
  - Very general-purpose
  - Limited inductive bias
- Stack of *feedforward* layers
  - Input layer, hidden layer(s), output layer
  - **Nonlinear** activation function
- Hierarchical representation

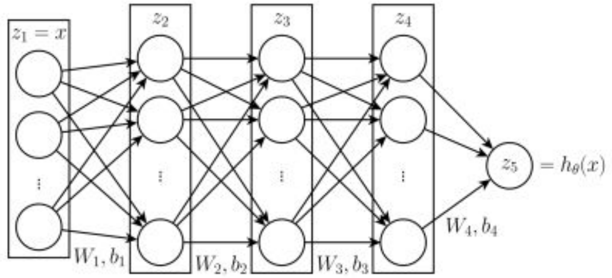
This is most of what you need for the homeworks (architecture-wise)!



Source: https://cs231n.github.io/neural-networks-1/

Multi-layer perceptrons (MLPs)

- Some notation
  - Input features: $x \in \mathbb{R}^d$
  - Outputs: $y \in \mathcal{Y}$
  - Network parameters: $\theta \in \mathbb{R}^k$
  - Network function: $h_\theta : \mathbb{R}^d \to \mathcal{Y}$
  - Activation function: $f \in \{\sigma, \tanh, \mathrm{relu}, \dots\}$
  - Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+ \in \{\mathrm{mse}, \mathrm{nll}, \mathrm{bce}, \dots\}$

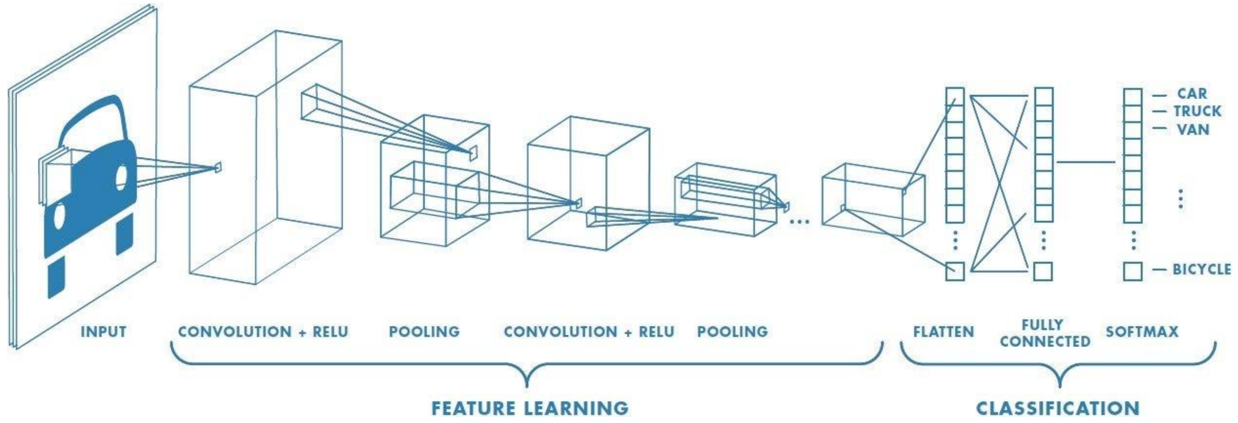


- Computing the network output

$$z_{i+1} = f_i(W_i z_i + b_i), \quad i = 1, \dots, k-1, \quad z_1 = x$$
$$h_\theta(x) = z_k$$

Source: 10-403 recitation slides, https://cmudeeprl.github.io/403website_s22/assets/lectures/s22/recitation_1-neural_nets.pdf

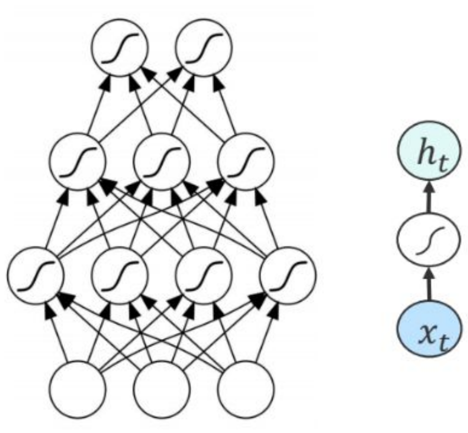
Convolutional Neural Networks



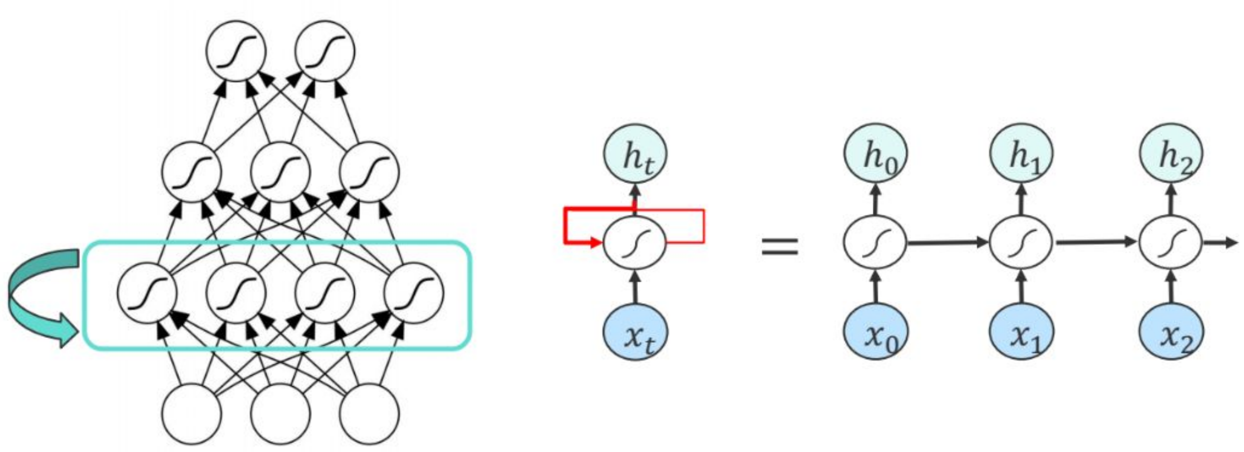
- A convolutional neural network can have several layers that **each learn to detect different features of an image**. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.
- Why not flatten an image and pass it through an MLP?
  - CNNs are able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters
  - CNNs are able to fit the image dataset better due to the reduction in the number of parameters and weights reusability

Source: A Comprehensive Guide to Convolutional Neural Networks (https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/)

Recurrent neural networks (RNNs)



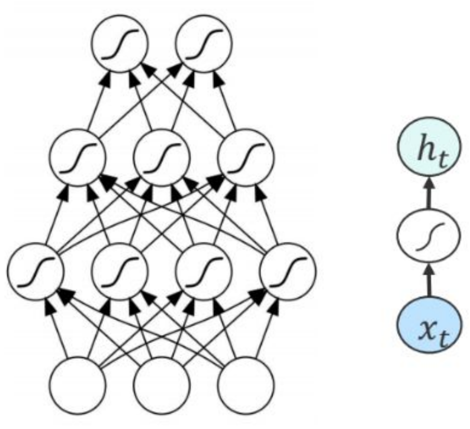
MLP: at each timestep, the computation is independent of previous timesteps (*stateless*)

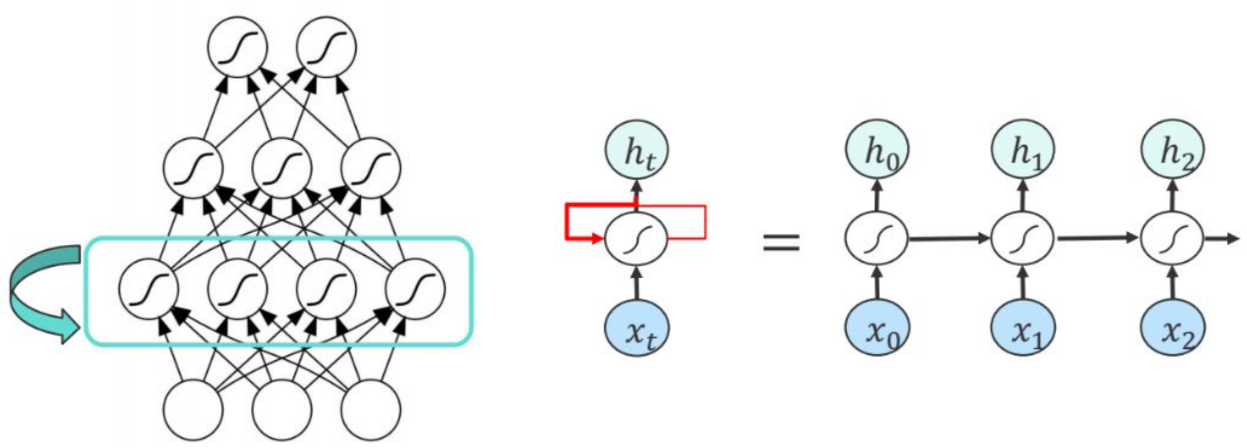


RNN: at each timestep, the computation is dependent on previous timesteps (*stateful*)

Source: Eric Xing, https://www.dropbox.com/s/uetn2j5opxjg0x8/Deep%20Sequence%20Models.pdf?dl=0

Recurrent neural networks (RNNs)





- Natural fit for sequential decision-making problems
- Vanishing / exploding gradient problem (alleviated by LSTMs / GRUs)
- Hard to parallelize due to sequential computation

Source: Eric Xing, https://www.dropbox.com/s/uetn2j5opxjg0x8/Deep%20Sequence%20Models.pdf?dl=0

Attention

Basic building block used in transformer models

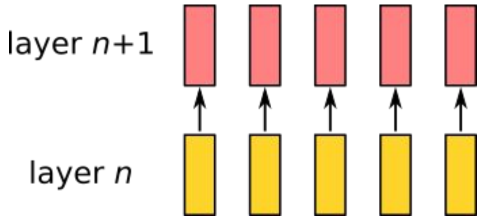
Can think of as a *differentiable dictionary lookup* that takes three arguments:

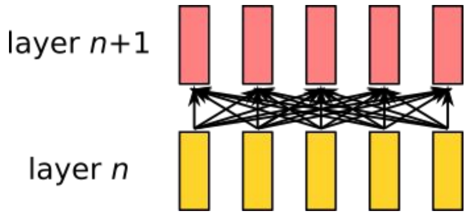
Query (**Q**), Key (**K**), Value (**V**)

Each argument (and output) is a sequence of vectors with same sequence length

Two main differences from standard feedforward layers:

1. Attention can selectively attend to / ignore specific sequence elements
   a. Standard feedforward layers can do this too, but the key difference is *sparsity*
2. Handles *arbitrary length* sequences of vectors





Dot-product attention

Inputs: a query q and a set of key-value (k-v) pairs to an output

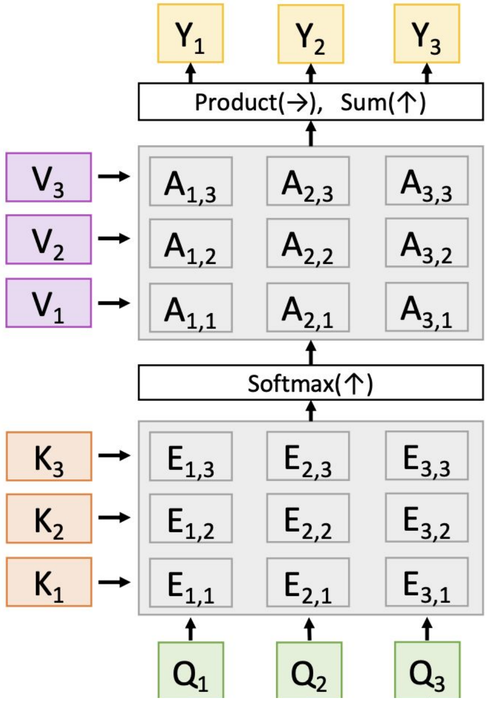
Query, keys, values, and output are all vectors

Output is weighted sum of values, where

Weight of each value is computed by an inner product of query and corresponding key

Queries and keys have same dimensionality d_k value have d_v

$$A(Q,K,V) = softmax(QK^T)V$$



Scaled dot-product attention

Inputs: a query q and a set of key-value (k-v) pairs to an output

Query, keys, values, and output are all vectors

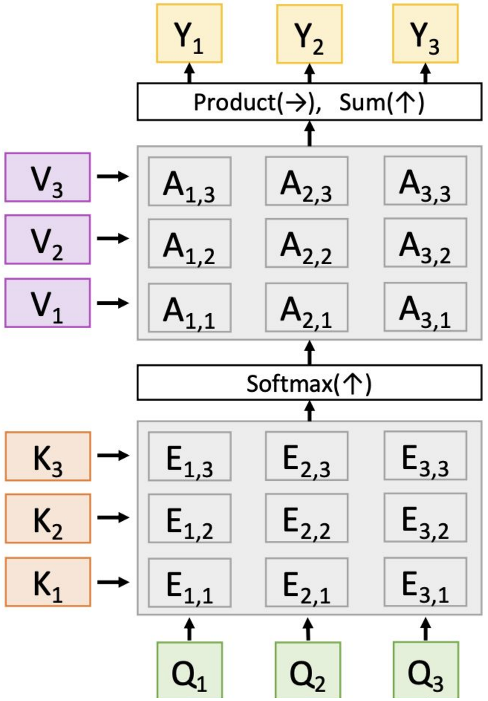
Output is weighted sum of values, where

Weight of each value is computed by an inner product of query and corresponding key

Queries and keys have same dimensionality d_k value have d_v

$$\cancel{A(Q,K,V) = softmax(QK^T)V}$$

$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Self-attention vs Cross-attention

- Where do Q, K, V come from?
    - Recall that Q is the query, K is the key, and V is the value (all are vectors)

- **Self-attention**: Q, K, V are all computed from the same input vector

- **Cross-attention**: Q comes from input vector v1, and (K,V) comes from another input vector v2 (we say v1 *cross-attends* to v2)

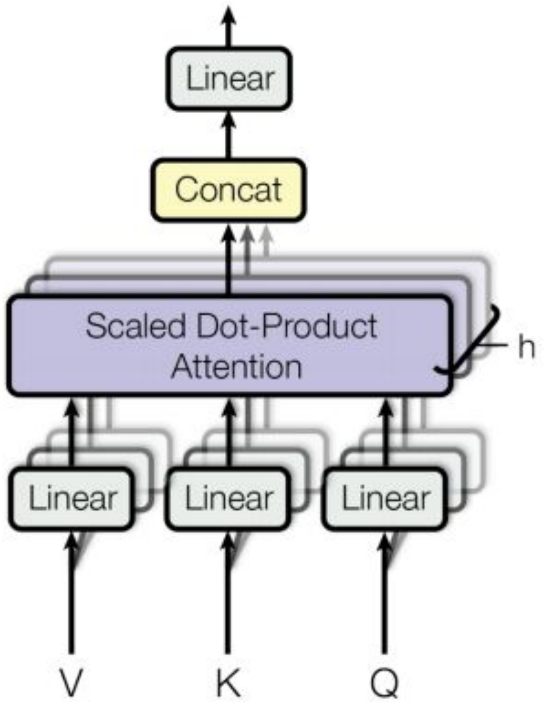
$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Source: Richard Socher, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf

Multi-headed attention

For an input vector, we can generate an arbitrary number of Q, K, V vectors

If we generate *h* sets of Q,K,V vectors, then we have multi-headed attention with *h* heads



Source: Richard Socher, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf

Transformers

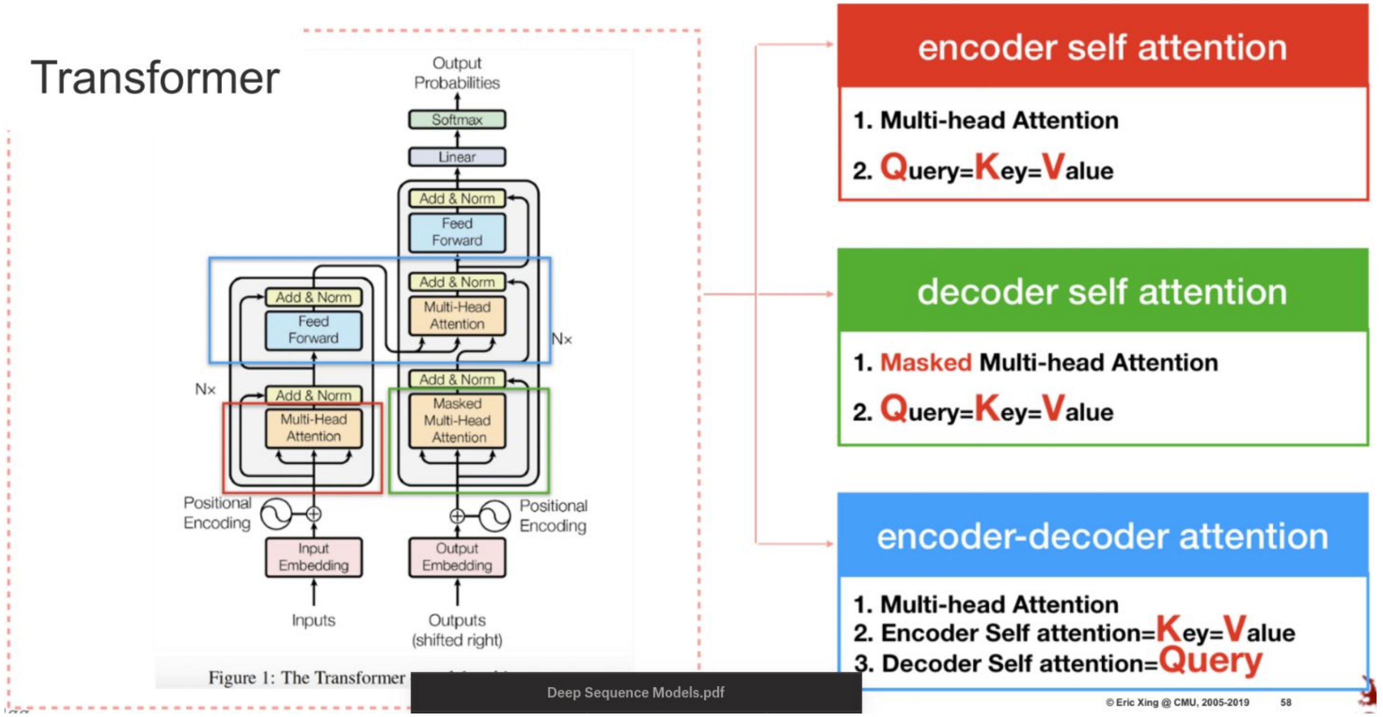


Figure 1: The Transformer

encoder self attention

1. Multi-head Attention
2. Query=Key=Value

decoder self attention

1. Masked Multi-head Attention
2. Query=Key=Value

encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query

Image Source: Eric Xing, Attention is all you need: https://arxiv.org/pdf/1706.03762.pdf

Types of Learning

1. Supervised Learning
   a. Labeled data
   b. Goal is to predict the label (i.e. classification or regression) given a new piece of data
2. Unsupervised Learning
   a. Unlabeled data
   b. Goal is to cluster/generate a new piece of data
3. Reinforcement Learning
   a. Given access to an environment
   b. Goal is to maximize the reward in that environment by taking actions
   c. You can formulate both SL and USL as a form of RL

PyTorch Code

This should mostly be review for you:

https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html

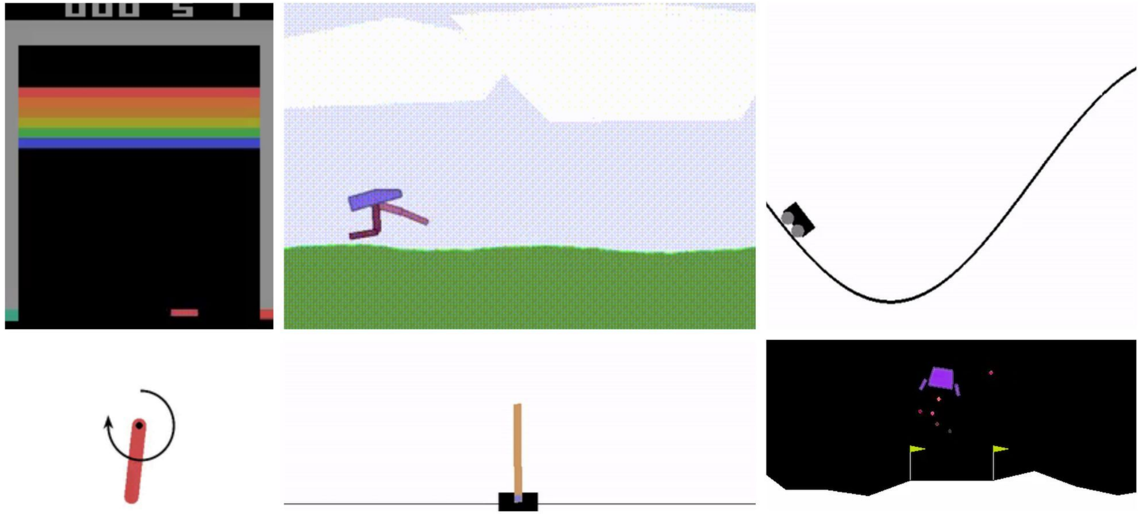
Tensorflow Code

This should also mostly be review for you:

https://www.tensorflow.org/tutorials/quickstart/advanced

Gymnasium

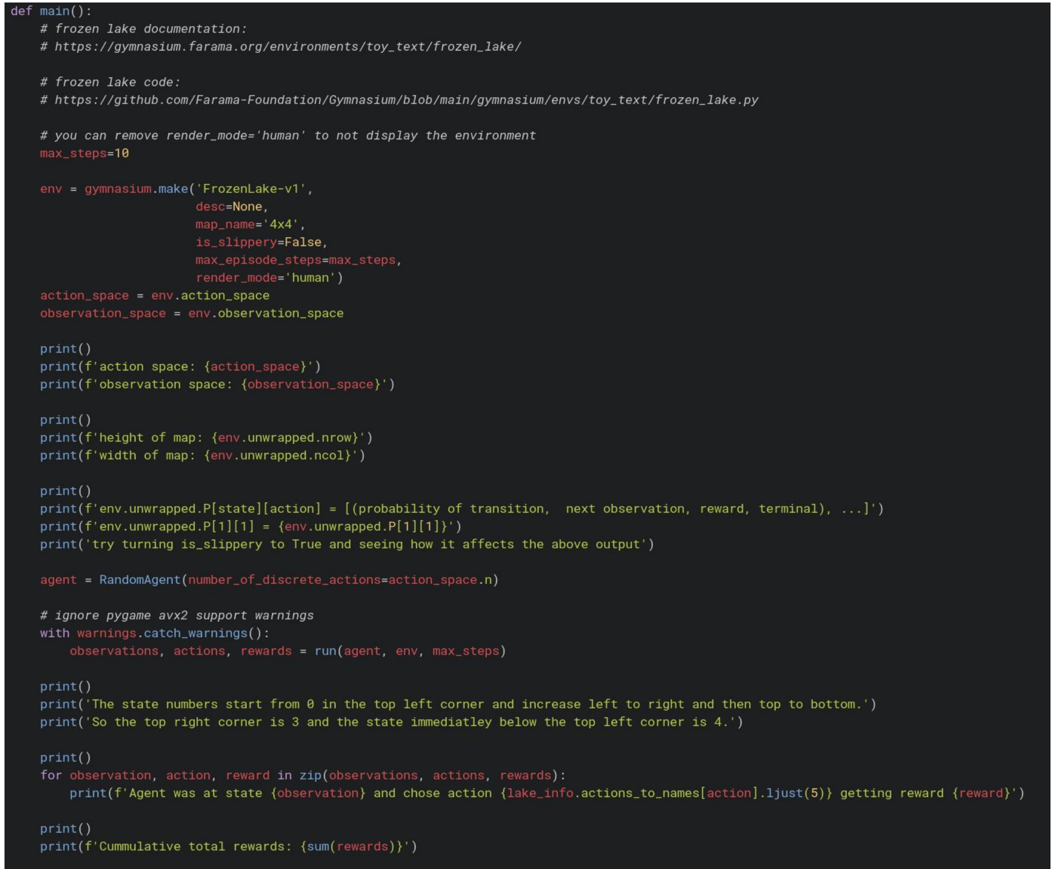
- Gymnasium is a Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments
- Many popular environments such as cartpole and mountain car can be used to evaluate the performance of an algorithm



Gymnasium Code

All of the following code will be included in your HW1 under HW1/code/frozen_lake/example.py

Gymnasium Code



```python
def main():
    # frozen lake documentation:
    # https://gymnasium.farama.org/environments/toy_text/frozen_lake/

    # frozen lake code:
    # https://github.com/Farama-Foundation/Gymnasium/blob/main/gymnasium/envs/toy_text/frozen_lake.py

    # you can remove render_mode='human' to not display the environment
    max_steps=10

    env = gymnasium.make('FrozenLake-v1',
                         desc=None,
                         map_name='4x4',
                         is_slippery=False,
                         max_episode_steps=max_steps,
                         render_mode='human')
    action_space = env.action_space
    observation_space = env.observation_space

    print()
    print(f'action space: {action_space}')
    print(f'observation space: {observation_space}')

    print()
    print(f'height of map: {env.unwrapped.nrow}')
    print(f'width of map: {env.unwrapped.ncol}')

    print()
    print(f'env.unwrapped.P[state][action] = [(probability of transition,  next observation, reward, terminal), ...]')
    print(f'env.unwrapped.P[1][1] = {env.unwrapped.P[1][1]}')
    print('try turning is_slippery to True and seeing how it affects the above output')

    agent = RandomAgent(number_of_discrete_actions=action_space.n)

    # ignore pygame avx2 support warnings
    with warnings.catch_warnings():
        observations, actions, rewards = run(agent, env, max_steps)

    print()
    print('The state numbers start from 0 in the top left corner and increase left to right and then top to bottom.')
    print('So the top right corner is 3 and the state immediatley below the top left corner is 4.')

    print()
    for observation, action, reward in zip(observations, actions, rewards):
        print(f'Agent was at state {observation} and chose action {lake_info.actions_to_names[action].ljust(5)} getting reward {reward}')

    print()
    print(f'Cummulative total rewards: {sum(rewards)}')
```

Gymnasium Code

```python
def run(agent, env, max_steps):
    observation, info = env.reset()
    episode_observations, episode_actions, episode_rewards = [], [], []

    for _ in range(max_steps):
        action = agent.get_action(observation)

        next_observation, reward, terminated, truncated, info = env.step(action)

        episode_observations.append(observation)
        episode_actions.append(action)
        episode_rewards.append(reward)

        observation = next_observation

        if terminated:
            break

    return(episode_observations, episode_actions, episode_rewards)
```

Gymnasium Demo

https://colab.research.google.com/drive/10PLRkBJTYYzuqAxNtxdhNWyjb2VZvdCt

Questions?