Q1 Answer ALL parts. Total marks awarded for this question: 10 marks.

Part a) How is experience captured, processed, stored, and sampled for training a Deep Q Networks (DQN) on Atari? Illustrate the discussion with coding fragments or pseudocode.3 marks.

Answer:

1. Experience Capture: As the agent interacts with the Atari environment, it captures the state (typically a stack of the last four frames to capture motion), the action taken, the reward received, and the next state. Each of these elements together forms a tuple often referred to as an "experience".

2. Experience Storage: These experiences are stored in a data structure known as the "replay buffer". The replay buffer is essentially a rolling queue with a fixed maximum size; once it is full, older experiences are discarded to make room for new ones.

3. Experience Processing: Before being stored in the replay buffer, the raw pixel data from the Atari frames are often preprocessed. This preprocessing typically includes resizing the frames, converting them to grayscale to reduce dimensionality, and normalizing pixel values.

4. Experience Sampling: When the network is updated (training step), a mini-batch of experiences is sampled randomly from the replay buffer. This random sampling helps to break the correlation between consecutive samples, which is crucial for the stability of the learning Algorithms.

```
class DQN_Agent:
def __init__(self, state_size, action_size, replay_buffer_size):
        self.state_size = state_size
        self.action_size = action_size
        self.replay_buffer = deque(maxlen=replay_buffer_size)
        self.model = build_network(state_size, action_size) # Neural network

def preprocess_state(self, state):
        # Convert state to grayscale and resize
```

```
        processed_state = resize_to(state, (84, 84))
        processed_state = convert_to_grayscale(processed_state)
        return normalize(processed_state)
def store_experience(self, state, action, reward, next_state, done):
        self.replay_buffer.append((state, action, reward, next_state, done))
        def sample_experiences(self, batch_size):
        return random.sample(self.replay_buffer, batch_size)

def train(self, batch_size):
        mini_batch = self.sample_experiences(batch_size)
        # Update the network based on experiences
        for state, action, reward, next_state, done in mini_batch:
                target = reward + 0.99 * np.max(self.model.predict(next_state)) * (1 - done)
                target_f = self.model.predict(state)
                target_f[0][action] = target
                self.model.fit(state, target_f, epochs=1, verbose=0)
```

Part b) Describe the target used to train a DQN. Illustrate the discussion with coding fragments or pseudocode. 3 marks.

Answer:

Part c) What is the cause of maximisation bias in DQNs? Describe an approach that can be used to reduce maximisation bias. (4 marks)

Answer:

In the context of Deep Q-Networks (DQNs), **maximization bias** refers to the overestimation of action values due to the max operator in the Q-learning update rule, which chooses the maximum estimated future reward. This can lead to consistently biased high evaluations of certain actions that are not actually optimal.

Example of Maximization Bias: Consider a DQN trained in a gaming environment where an agent can choose between two actions at each state, both leading to similar rewards on average. However,due to random chance, early action choices might return slightly higher rewards for one action. Using the traditional Q-learning update rule, the model might start to overestimate the value of this action because it always selects the maximum future reward for updates, thus reinforcing the bias.

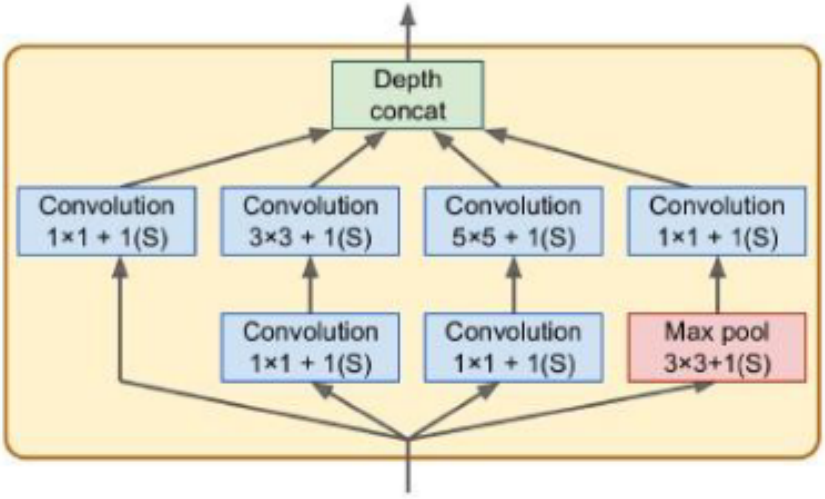
Two Approaches to Reduce Maximization Bias:

1. **Double Q-Learning:** This method uses two separate Q-value estimators to reduce bias. One estimator is used to select the best action, and the other to evaluate its value. By decoupling the action selection from its value estimation, it prevents the overoptimistic evaluation typical in standard Q-learning.
2. **Clipped Double Q-Learning:** Often used in Twin Delayed Deep Deterministic (TD3) policy gradient methods, this approach involves maintaining two separate estimators and updating each using the minimum estimate from both for the next state's value. This conservative estimation approach helps in further reducing the overestimations seen in Double Q-learning

Q2 Answer ALL parts. Total marks awarded for this question: 10 marks.

Part a) Explain why the number of parameters in GoogleLeNet using Inception Modules are significantly less than AlexNet - 6 million as opposed to 60 million.The answer should focus on the Inception module. Illustrate the answer with diagrams and/or rough calculations.3 marks.

Answer



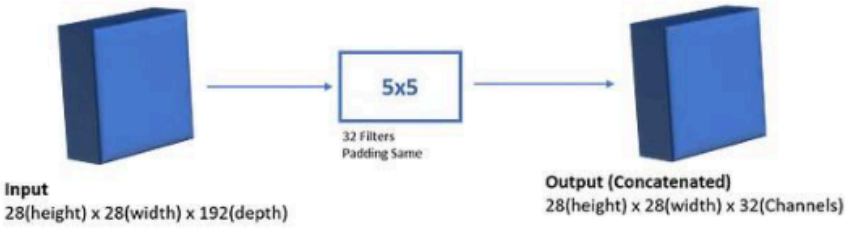


Figure 5. A naive filter.

- Num ops: multiply the number of outputs that are required to be provided (28x28x32), with the number of multipliers needed to work out a single value within the output (5x5x192).
- Num multiplier ops = (output dimensions) * (filter dimensions) * (depth of input channel)
- Num multiplier ops = (28 x 28 x 32) x (5 x 5) x (192)
- = 120, 244, 400

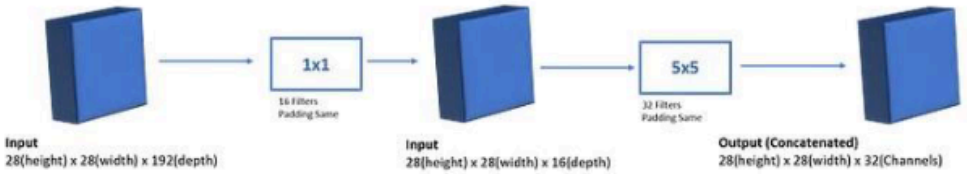


Figure 6. reducing computation using a 1x1 filter

- Num multiplication ops = ((28 x 28 x 16) * (1 x 1 x 192)) + ((28 x 28 x 32) * (5 x 5 x 15))
- Num multiplication ops =(2,408,448)+ (10,976,000)
- = 12,443,648.

1x1 kernel with reduced number of filters reduces spatial dimensions but keeps depth information.

• Although they cannot capture patterns along the spatial dimension, they can capture patterns across depth dimension (i.e., across channels).
• They are configured to output fewer feature maps than their inputs, so they serve as bottleneck layers, meaning they reduce dimensionality. This cuts the computational cost and the number of parameters, speeding up training and improving generalization.
• Each pair of convolutional layers ([1 × 1, 3 × 3] and [1 × 1, 5 × 5]) acts like a single powerful convolutional layer, capable of capturing more complex patterns. A convolutional layer is equivalent to sweeping a dense layer across the image (at each location, it only looks at a small receptive field), and these pairs of convolutional layers are equivalent to sweeping two-layer neural network across the image.
[Geron 2018] Page 5 of 6

Benefits of the Inception Module
• High-performance gain on convolutional neural networks
• Efficient utilisation of computing resource with minimal increase in computation load for the high-performance output of an Inception network.
• Ability to extract features from input data at varying scales through the utilisation of varying

convolutional filter sizes.
• 1x1 conv filters learn cross channels patterns, which contributes to the overall feature extractions
capabilities of the network.

Part b) Ioffe and Szegedy (2015) proposed Batch Normalisation as a mechanism to reduce theimpact of vanishing gradients. How many parameters in the three Batch Normalisation layers in Figure Q2? Of these, how many are trainable? Please show the calculations.

```
L1. model = keras.models.Sequential([
L2. keras.layers.Flatten(input_shape=[28, 28]),
L3. keras.layers.BatchNormalization(),
L4. keras.layers.Dense(150,activation="relu", ernel_initializer=“he_normal"),
L5. keras.layers.BatchNormalization(),
L6. keras.layers.Dense(100, activation="relu", kernel_initializer=“he_normal"),
L7. keras.layers.BatchNormalization(),
L8. keras.layers.Dense(10, activation="softmax")])
```

Figure Q2
3 marks.

Answer:

1. First BatchNormalization Layer (L3)

- The input to this layer is the output of a Flatten layer, which simply reshapes the 28x28 input images into a 784-dimensional vector (since 28*28 = 784).
- Number of parameters in a BN layer = 2 * size of input feature vector (for γ and β each) •
- Therefore, parameters = 2 * 784 = 1568

2. Second BatchNormalization Layer (L5)

- The input to this layer comes from a Dense layer with 150 units.
- Number of parameters = 2 * 150 = 300

3. Third BatchNormalization Layer (L7)

- The input to this layer comes from another Dense layer with 100 units.
- Number of parameters = 2 * 100 = 200

Total Parameters in Batch Normalization Layers

- Total parameters in all BN layers = 1568 (L3) + 300 (L5) + 200 (L7) = 2068

Part c) Describe the key concept(s) in ResNet. Include a discussion on the purpose of a kernel of size 1x1 with stride 2. Illustrate the discussion with a diagram

Answer:

Residual Networks (ResNets) are a class of deep neural networks that were introduced by Kaiming He et al. in 2015. They are primarily known for their ability to train very deep networks by using residual blocks that incorporate shortcut connections (also known as skip connections).

These connections allow gradients to flow through the network more effectively, addressing the vanishing gradient problem and enabling the training of networks with much greater depth than was previously feasible.

Residual Blocks and Skip Connections
The core concept of ResNet is the residual block. Each block contains two main paths. The **first path** has the conventional layers — typically convolutional layers, batch normalization, and ReLU
activations. The **second path** is a shortcut connection that skips these layers and connects the input of the block to its output.

The output of a residual block is calculated as the element-wise addition of the output from the conventional path and the shortcut path:

Output=Activation (Convolutional Output+Shortcut Input)

The shortcut connections help mitigate the vanishing gradient problem by allowing an alternative path for the gradient during backpropagation. This setup makes it possible to train networks that are significantly deeper than those used previously.

Skip connections speed up learning.

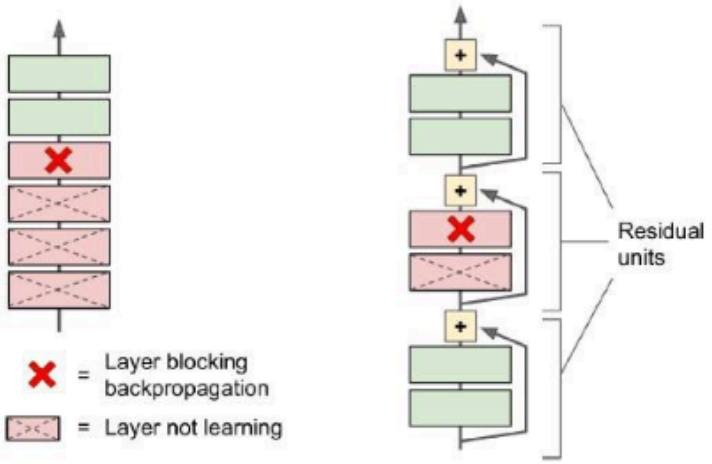


Figure 7: Speeding up learning in ResNet through Skip connections.

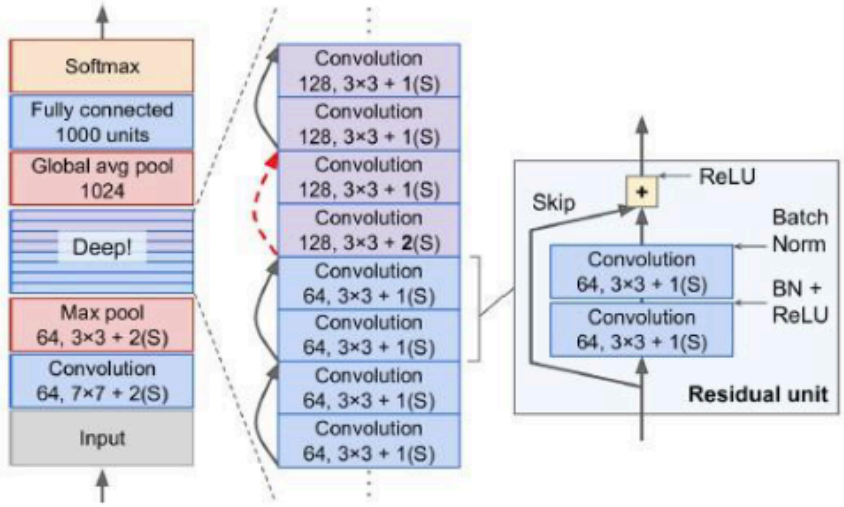


Figure 8. ResNet

1x1 Convolution with Stride 2

A 1x1 convolution kernel with a stride of 2 serves a specific purpose in the architecture of ResNets, particularly in the context of dimensionality reduction and channel adjustment:

- **Dimensionality Reduction**: When increasing the depth of the network, it is often necessary to reduce the spatial dimensions of the feature maps (height and width) to control the computational complexity. Using a stride of 2 in a 1x1 convolution effectively reduces the dimensionality of the feature maps by half in each spatial dimension.
- **Channel Adjustment:** The 1x1 convolution also adjusts the number of channels in the feature maps. This is useful when the number of input channels needs to be increased or decreased to match the output channels of a residual block, facilitating the element-wise addition in the shortcut connection.

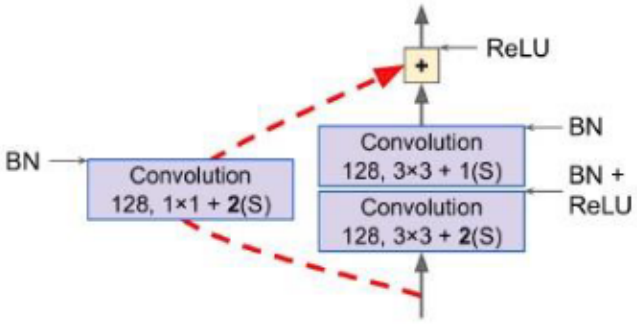


Figure 9. Using a 1x1 filter with stride 2 when dimensionality changes in ResNet stack.

Q3 Answer ALL parts. Total marks awarded for this question: 10 marks.

Part a) List the steps in the REINFORCE Algorithm. 3 marks.

Answer:

The REINFORCE algorithm, introduced by Ronald J. Williams in 1992, is a popular policy gradient method for training reinforcement learning agents. Here are the steps involved in the

REINFORCE algorithm:

1. Initialize Policy Parameters: Initialize the parameters of the **policy function** $\pi\theta(a|s)$, where

- θ represents the weights of the neural network or other function approximator used to represent the policy.

2. Generate Trajectories: Interact with the environment to generate trajectories (sequences of states, actions, and rewards) by following the **current policy** $\pi\theta$. This involves executing actions in the environment and observing the resulting rewards and next states.

3. Compute Returns:
For each time step t in the trajectory,
compute the return Gt, which is the cumulative sum of rewards from time t until the end of the trajectory.

The return at time t can be computed as the sum of discounted rewards:

$$Gt = \sum k = tT \gamma k - tRk$$

where

- T is the time step at the end of the trajectory,
- $Rk$is the reward at time step k, and
- γ is the discount factor.

4. Update Policy Parameters: Update the policy parameters θ using the gradient of the expected return with respect to the policy parameters.

This gradient is estimated using the following formula:

$$\nabla \theta J(\theta) = E\pi\theta \ [\nabla \theta \ \textbf{log} \ \pi\theta(a|s) \ Gt]$$

Where

- $\nabla \theta J(\theta)$ is the gradient of the expected return
- $J(\theta)$ with respect to the policy parameters θ,
- $E\pi\theta$ denotes the expectation under the **current policy** $\pi\theta$**.**

5. **Update Policy Parameters using Gradient Ascent:**
Perform a gradient ascent update to adjust the policy parameters in the direction that maximizes the expected return:

$$\theta \leftarrow \theta + \alpha \nabla \theta J(\theta)$$ where α **is the learning rate.**

6. **Repeat**: Repeat steps 2-5 for multiple episodes or until convergence, gradually improving the policy parameters to maximize the expected return.

The REINFORCE algorithm is a foundational policy gradient method that forms the basis for more advanced reinforcement learning algorithms. It directly learns the policy by estimating gradients of the expected return and updating the policy parameters accordingly, making it applicable to a wide range of reinforcement learning problems.

Part b) Describe three advantages of Policy Gradient approaches. 3 marks.

Answer:

1. **Capability to Learn Stochastic Policies**: These methods can learn policies that select actions probabilistically, which is useful in environments where variability in action can lead to better overall strategies.
2. **Better Convergence Properties:** Unlike value-based methods, which can suffer from unstable learning progress, Policy Gradient methods update policies in a direction that is

mathematically guaranteed to improve performance, leading to smoother and more reliable Learning.
3. **Effectiveness in High-Dimensional Action Spaces:** Policy Gradient methods are particularly well-suited for tasks with continuous or very large action spaces, such as robotics, because they can naturally operate without needing to discretize the action space.

Part c) The coding fragment is Figure Q3 is an excerpt of an implementation of the REINFORCE method for the cartpole. Provide a detailed explanation for lines 2 to 11. (4 marks.)

```
L1   for iterations in range(n_Iterations):
L2       all_rewards, all_grads = play_multiple_episodes(env, n_episodes_per_update,
                n_max_steps, model, loss_fn)
L3       total_rewards = sum(map(sum, all_rewards))
L4       all_final_rewards = discount_and_normalise_rewards(all_rewards, discount_rate)
L5       all_mean_grads = []
L6       for var_index in range(len(model.trainable_variables)):
L7           mean_grads = tf.reduce.mean([final_reward * all_grads[episode_index][step][var_index]
L8               for episode_index, final_rewards in enumerate(all_final_rewards)
L9                   for step, final_reward in enumerate(final_rewards)], axis=0)
L10          all_mean_grads.append(mean_grads)
L11      optimizer.apply_gradients(zip(all_mean_grads, models.trainable_variables))
```

Figure Q3. Adapted from Maxim Lapan. Deep Reinforcement Learning. 2018. Packt Publications.

Answer:

The provided code snippet outlines the implementation of the REINFORCE algorithm in a reinforcement learning setting, specifically for training an agent like the cartpole. Here's a detailed breakdown of each line from L2 to L11:

Line 2:

```
L2       all_rewards, all_grads = play_multiple_episodes(env, n_episodes_per_update,
             n_max_steps, model, loss_fn)
```

- **Functionality**: This line calls a function `play_multiple_episodes`, which plays multiple episodes of the environment `env`.
- **Parameters**:
  - `**n_episodes_per_update**`: The number of episodes to run before updating the model.
  - `**n_max_steps**`: Maximum steps per episode to prevent endlessly running episodes.
  - `**model**`: Neural network model used to make predictions (actions).
  - `**loss_fn**`: Loss function used to compute the error during training.

- **Returns**:
  - `**all_rewards**`: A list containing the rewards obtained in each episode.
  - `**all_grads**`: A list of gradients of the model's parameters with respect to the loss function for each action taken at each step of each episode.

Line 3: total_rewards = sum(map(sum, all_rewards))
- **Functionality**: Calculates the total rewards accumulated over all episodes by summing up all the rewards for each step in each episode.

Line 4:
all_final_rewards = discount_and_normalise_rewards(all_rewards, discount_rate)

- **Functionality**: This function adjusts the rewards:
  - **Discounting**: Applies a discount factor (`discount_rate`) to the rewards, which reduces the importance of rewards received at later timesteps.
  - **Normalisation**: Normalises these discounted rewards across all episodes to reduce variance and improve convergence.

Line 5: all_mean_grads = []

- **Purpose**: Initializes an empty list to store the mean gradients computed for each trainable variable across all episodes.

Lines 6-11

```
L6    for var_index in range(len(model.trainable_variables)):
L7      mean_grads = tf.reduce.mean([final_reward * all_grads[episode_index][step][var_index]
L8          for episode_index, final_rewards in enumerate(all_final_rewards)
L9              for step, final_reward in enumerate(final_rewards)], axis=0)
L10     all_mean_grads.append(mean_grads)
L11   optimizer.apply_gradients(zip(all_mean_grads, models.trainable_variables))
```

- ****Lines 6-10****: Iterates over each trainable variable in the model. For each variable, it computes the mean of the product of the final adjusted reward and the gradients with respect to that variable across all episodes and steps. This results in an average gradient that takes into account how good the actions were in terms of future rewards.
- **Line 11**:
  - **`optimizer.apply_gradients()`**: Applies the computed mean gradients to the model's trainable variables to update them. The `zip` function pairs each mean gradient with its corresponding variable.
  - **Purpose**: This step effectively updates the model parameters in the direction that increases the likelihood of actions leading to higher rewards, following the policy gradient update rule.

This series of operations allows the REINFORCE algorithm to learn policies that maximize the cumulative rewards by adjusting the model parameters using gradient ascent on expected rewards.

Q4

Part A - Briefly describe the Physical Symbol System Hypothesis (PSSH). Compare and contrast the PSSH with Machine Learning

Answer:

The Physical Symbol System Hypothesis (PSSH) is a central theory in the field of artificial intelligence, proposed by Allen Newell and Herbert A. Simon in 1976. According to the PSSH, **a physical symbol system (such as a computer running software programs) has the necessary and sufficient means for general intelligent action.** The hypothesis posits that the manipulation of symbols (which are physical patterns) can be constructed into any expression or computation, including reasoning and problem-solving activities. The physical symbol system is capable of representing and manipulating symbols to produce new arrangements of symbols and can effectively simulate any act of human cognition.

Comparison with Machine Learning:*
1. Foundation and Focus:
-PSSH: Focuses on symbolic reasoning and manipulation where intelligence emerges from the ability to form, manipulate, and interpret symbols and symbolic structures. It emphasizes rule-based processing and logic.
- **Machine Learning (ML):** Focuses on the ability of systems to learn from data, identify patterns, and make decisions with minimal human intervention. ML uses statistical methods and algorithms to perform tasks without explicit programming for each specific task.

2. Method of Operation:
- **PSSH**: Operates through explicit rules and symbolic manipulation. It is deterministic and interpretable, relying heavily on crafted rules and logical structures.
- **ML**:** Typically operates through statistical generalization. Instead of using predefined rules, it adjusts its parameters based on feedback from the performance on data. It can be non-deterministic and sometimes offers less interpretability (especially in models like deep neural networks).

3. **Flexibility and Adaptability:**
- **PSSH**:** While highly logical and interpretable, it can be rigid, requiring manual adjustments or reprogramming to handle new types of problems or adapt to changes in the environment.

- **ML**:** Highly adaptable, as it can learn from new data. Models can improve or adapt their behavior over time based on the exposure to new data and experiences, making them more flexible in dynamic environments.

4. **Scope and Application:**
- **PSSH**:** Best suited for environments where rules and relationships are well-defined and can be explicitly encoded, such as in expert systems.
- **ML**:** More effective in complex, real-world scenarios where patterns and relationships are not explicitly known beforehand or are too complex to be manually encoded, such as image recognition, natural language processing, and predictive analytics.

In summary, the Physical Symbol System Hypothesis and Machine Learning represent two different paradigms within artificial intelligence. PSSH focuses on symbolic, rule-based intelligence, highly structured and interpretable, while Machine Learning is centered around learning from data, adaptable, and often more suited to handling unstructured or complex data scenarios. Both approaches have their strengths and are often complementary in sophisticated AI systems.

Part B - Compare and contrast Evolutionary Computation with Reinforcement Learning (sutto and barto initial pages)

Answer:

Evolutionary Computation (EC) and Reinforcement Learning (RL) represent two distinct paradigms in artificial intelligence, each based on different inspirations and methodologies.

EC is inspired by biological evolution, using mechanisms such as natural selection and mutation to evolve solutions over generations. It's particularly suited for complex optimization problems where solutions evolve in a population-based approach.

RL is inspired by behaviorist psychology and focuses on learning through interaction with an environment to maximize cumulative rewards. It is effective in decision-making tasks where an agent learns from actions' consequences to develop policies for optimal behavior.

While **EC** explores a broad solution space with genetic operations across populations, **RL** incrementally adjusts actions in a state-based manner through trial and error, making it highly responsive to dynamic environments but sensitive to reward structures.

These differences highlight their suitability to varied types of problems, with EC being favored in global search scenarios and RL in situations requiring adaptive decision-making strategies.

Q5 Answer ALL parts. Total marks awarded for this question: 10 marks.

Part a) What is Reinforcement Learning? What are the key issues that an RL agent must address? (3 marks.)

Answer:

Reinforcement Learning (RL) Overview
Reinforcement Learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. In this learning paradigm, the agent takes actions in a given state of the environment, which in turn may result in the agent receiving a reward or penalty and transitioning to a new state. The goal of the agent is to learn a policy — a strategy of choosing actions in given states — that maximizes the cumulative reward it receives over time.

Key Issues in Reinforcement Learning
1. Exploration vs. Exploitation:

• **Exploration** involves the agent trying new actions to discover their effects and to learn more about the environment. It's crucial for finding potentially better strategies that have not been tried yet.
• **Exploitation** means using the knowledge the agent has already gained to make decisions that maximize the reward. It focuses on leveraging the best strategy known so far.
• Balancing these two aspects is a central challenge in RL. Too much exploration can lead to suboptimal performance in the short term, while too much exploitation might prevent the agent from finding the most rewarding strategies.

2. Credit Assignment Problem:
• Determining which actions are responsible for obtaining a particular reward can be difficult, especially when rewards are delayed. This is known as the "credit assignment problem."
• The agent needs to figure out which actions in a sequence were crucial for achieving success and how to appropriately update the policy based on rewards received possibly much later than the actions taken.

3. Dimensionality of State and Action Spaces:
• Many RL problems involve very large or continuous state spaces and action spaces, making them computationally challenging to handle.
• Discretization of continuous spaces, function approximation techniques (like neural networks in Deep RL), and policy gradient methods are some approaches to deal with high-dimensional spaces

4. Partial Observability:

• In many real-world environments, the agent might not have access to the complete state of the environment but only a partial view (partial observability).
• This scenario complicates the learning process, as the agent must make decisions with incomplete information. Techniques like using recurrent neural networks or maintaining a belief about the state of the environment are approaches to address this Issue.

5. Stability and Convergence:
• Learning stable and converging policies in RL can be difficult, especially with function approximators in the loop (e.g., neural networks in Deep RL).
• The interaction between the policy being learned and the data it generates can lead to non-stationarities and potentially to divergent behavior. Ensuring stability and convergence often requires careful tuning of the learning parameters and algorithm design.

6. Generalization Across States:
• An agent trained in a specific environment might overfit to that environment and perform poorly in slightly different settings.
• Generalization involves the ability of an RL agent to perform well not just in the conditions it was trained under but also in new, unseen environments. Techniques such as regularization, domain randomization, and meta-learning are employed to improve generalization.

Part b) The path computed by the Q learning algorithm in Figure-Q5 is adjacent to the cliff edge, with the Sarsa path being further back. Draw a plot of the expected average rewards for Q and Sarsa. Explain why Sarsa computes a “safer” path when compared to Q learning. (3 marks.)

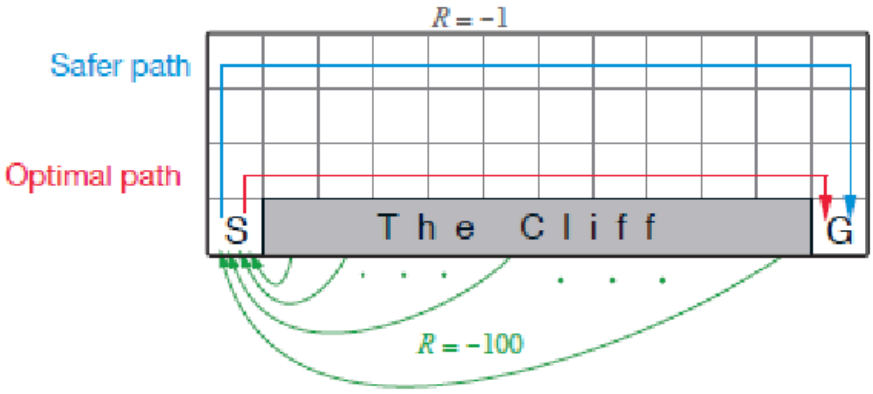


Figure Q5. Adapted from Sutton and Barto, Reinforcement Learning, 2nd Ed. 2018. MIT Press.

Answer:

c) What is meant by the terms on-policy and off policy in the context of TD methods? The discussion should include the equations for an on policy and off policy update.(4 marks.)

Answer:

In the context of Temporal Difference (TD) learning methods in reinforcement learning, the terms on-policy and off-policy describe two distinct approaches to the way learning and action selection are handled during the training process. These terms determine whether the same policy that is being evaluated and improved is also used to make decisions, or if a different policy is used for decision-making.

On-Policy Methods:
On-policy methods evaluate or improve the policy that is also used to make decisions about actions. Essentially, the learning policy (the policy that dictates what actions to take) and the target policy (the policy that is being improved) are the same.

SARSA (State-Action-Reward-State-Action) is a classic example of an on-policy TD method. The update formula for SARSA, which reflects this on-policy characteristic, is:

$$Q(St,At) \leftarrow Q(St,At) + \alpha[Rt+1+\gamma Q(St+1,At+1) - Q(St,At)]$$

- $Q(St,At)$ is the current estimate of the state-action value.
- α is the learning rate.
- $Rt+1$ is the reward received after taking action AtAt in state StSt.
- γ is the discount factor.
- $Q(St+1,At+1)$ represents the estimated value for the next state and action, where $At+1$ is chosen using the same policy that determined At.

Off-Policy Methods:
Off-policy methods, on the other hand, evaluate or improve a policy that is different from the policy used to generate the data. This approach allows the agent to learn from actions that are outside the current policy, potentially exploring and learning from a broader range of experiences.

Q-learning is a well-known off-policy TD method. Its update formula is:

$$Q(St,At) \leftarrow Q(St,At) + \alpha[Rt+1+\gamma \max_a Q(St+1,a) - Q(St,At)]$$

- $Q(St,At)$ is the current estimate of the state-action value.
- α is the learning rate.
- $Rt+1$ is the reward received after taking action AtAt in state StSt.
- γ is the discount factor.
- **max** $aQ(St+1,a)$ represents the maximum estimated value for the next state across all possible actions, reflecting the best possible decision according to the learned Q- values, not necessarily following the behavior policy.

Summary:
The key difference lies in how the future action $At+1$ is selected:
• **On-Policy (SARSA):** The action At+1 is selected by the **same policy** that is being evaluated. This approach maintains consistency between the policy's actions during both learning and deployment.
• **Off-Policy (Q-Learning):** The action At+1 is chosen as the one that **maximizes** the state-action value regardless of the action selection policy used during data generation. This method allows the policy to learn from hypothetical optimal decisions, even if they are not performed during the policy's execution.

These approaches offer different advantages: on-policy methods ensure that the policy remains practical and closely tied to the actual performance of the agent, while off-policy methods allow for greater flexibility and potentially faster convergence by learning from a broader set of experiences

—----------

Q1 Answer ALL parts. Total marks awarded for this question: 10 marks.

a) How is experience captured, processed, stored, and sampled for training a Deep Q Networks (DQN) on Atari? Illustrate the discussion with coding fragments or pseudocode. (3 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

b) What is maximisation bias in the context of DQNs? Provide an example to illustrate the discussion. Describe two approaches that can be used to reduce maximisation bias (3 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

c) Describe the loss function used to train a DQN. Illustrate the discussion with coding fragments or pseudocode. (4 marks.)

Answer:

DQN Atari: The Loss Function

8

- Parameters in the network at iteration i are θ_i
- Optimal target → $r+\gamma\, max_{a'} Q^*(s', a')$
- Substituted by approximate target → $r+\gamma\, max_{a'} Q(s', a'; \theta_i^-)$
- where θ_i^- are the parameters from a previous iteration used to compute the target at iteration i
- The Loss function
- $L_i(\theta_i) = \mathrm{E}_{(s,a,r,s')}\left[\left(r + \gamma\, max_{a'} Q(s', a'; \theta_i^-) - Q(s, a, ; \theta_i)\right)^2\right]$ where $\theta_i^- = \theta_{i-1}$
- The Q update can be achieved by updating weights after every iteration using single samples and optimising L using stochastic gradient descent

```
loss = agent.model.train_on_batch(states, targets_full)
total_loss += loss
```

DQN Loss Function Details:

The specific formulation of the loss function for a DQN is:

$$L(\theta) = \mathbb{E}\left[\left(y^{TD} - Q(s, a; \theta)\right)^2\right]$$

where:

- θ represents the parameters of the Q-network.
- $Q(s, a; \theta)$ is the network's prediction of the Q-value for choosing action a in state s.
- y^{TD} is the TD target for the Q-value, defined as:
  $y^{TD} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$
  - r is the immediate reward received.
  - γ is the discount factor, which moderates the importance of future rewards.
  - s' is the next state following the action.
  - $\max_{a'} Q(s', a'; \theta^-)$ is the maximum predicted Q-value for the next state s', using a separate set of parameters θ^- from a target network, which helps to stabilize the updates.

Q2 Answer ALL parts. Total marks awarded for this question: 10 marks.

a) Explain why the number of parameters in GoogleLeNet using Inception modules is significantly less than AlexNet - 6 million as opposed to 60 million. The answer should focus exclusively on the Inception module. Illustrate the answer with a diagram and/or rough calculations. (3 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

b) What is a vanishing gradient? Briefly describe the cause(s) of this phenomenon. Describe technique(s) can be used to reduce the likelihood of vanishing gradients. Include diagram(s) if discussing activation functions. (3 marks.)

Answer

Vanishing Gradient Problem
The vanishing gradient problem is a challenge encountered in training deep neural networks, particularly those with many layers. This issue occurs when the gradients of the network's weights, calculated during backpropagation, become extremely small, effectively preventing the weights from changing their values, which stalls the training process.

Causes of Vanishing Gradients

1. Activation Functions: Traditional activation functions like the sigmoid or tanh function are primary culprits. These functions squeeze their input into a narrow output range in which the gradient can be very small. For example, in both sigmoid and tanh functions, the gradient approaches zero as the input moves away from zero. This small gradient is propagated backward through the network, and with each layer, it gets multiplied and shrinks exponentially, leading to very tiny gradients in earlier layers.

2. Deep Network Architecture: In very deep networks, as the gradient is back-propagated from the output towards the input layer, repeated multiplication may make the gradient infinitesimally small. As a result, initial layers learn very slowly, if at all, which hampers the overall learning process.

Techniques to Reduce Vanishing Gradients

1. Use of ReLU Activation Function:
• **ReLU (Rectified Linear Unit):** This activation function outputs the input directly if it is positive; otherwise, it outputs zero. It has become very popular because its derivative is either 0 (for negative inputs) or 1 (for positive inputs). This characteristic helps in mitigating the vanishing gradient problem as the gradient does not diminish when propagated through multiple layers.

Here is a simple diagram illustrating ReLU:

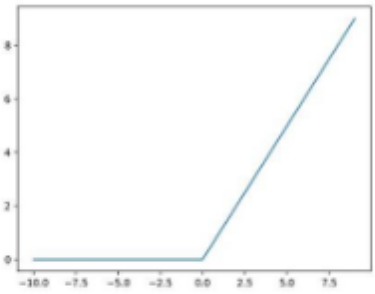


Figure 1. RELU

• **Leaky ReLU and Parametric ReLU:** These are variations of ReLU intended to allow a small, positive gradient when the unit is not active and the input is less than zero.

2. Proper Initialization:
• **He Initialization**: This technique sets the initial weights of the network by drawing from a distribution with a variance scaled by the inverse of the average number of neurons in the inputs and outputs. It's particularly useful when using ReLU activations, as it helps avoid small gradients at the start.

3. Batch Normalization:
• This method normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This normalization ensures that the values do not become too small or too large, thus maintaining a healthier gradient flow across deep networks.
These techniques help maintain a healthy gradient flow in deep networks, addressing the vanishing gradients problem and enabling more effective training of deep learning models.

c) Describe the key concept(s) in ResNet. Include a discussion on the purpose of a kernel of size 1x1 with stride 2. Illustrate the discussion with a diagram. (4 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

Q3 Answer ALL parts. Total marks awarded for this question: 10 marks.

a) What is Reinforcement Learning? What are the key issues that an RL agent must address? (3 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

b) Explain why it is stated that Temporal Difference (TD) methods bridge Dynamic Programming (DP) and Monte Carlo (MC) methods. (3 marks.)

Temporal Difference (TD) methods are often described as bridging the gap between Dynamic Programming (DP) methods and Monte Carlo (MC) methods in reinforcement learning. This is due to their unique characteristics that combine key features of both approaches, enabling them to leverage the strengths of each while addressing some of their respective limitations.

1. Relationship with Dynamic Programming (DP):
Dynamic Programming methods, such as value iteration and policy iteration, rely on a model of the environment. These methods compute the value function by using the Bellman equations, which require knowledge of the transition probabilities and rewards for all state-action pairs—a complete model of the environment. DP methods update the value function based on the expected next values, which are calculated using these probabilities.

TD Methods:

- Like DP, TD methods use the Bellman equations to update estimates of the value function.
- However, TD methods do not require a complete model of the environment. Instead, they update estimates based purely on the experiences sampled from the environment, using what is called bootstrapping. In bootstrapping, the current estimate of the value function is updated based on new empirical data plus existing estimates, akin to the recursive nature of DP but without needing a full model.

2. Relationship with Monte Carlo (MC) Methods:
Monte Carlo methods estimate the value function based on complete sequences of states and rewards, updating the value function only after the entire episode has been observed. This approach does not require a model of the environment and updates are based on actual returns rather than estimated returns.

TD Methods:

- TD methods also do not require a model and directly learn from empirical data, similar to MC methods.
- Unlike MC methods, which wait until the end of an episode to make updates (using the total actual return), TD methods update the value estimates incrementally after each step or a few steps within an episode. This update uses observed rewards and the current estimate of the value function for the next state, allowing TD methods to learn online and in a partially complete environment.

3. Bridging the Gap:
TD methods effectively bridge the gap between the reliance on a full model in DP and the complete reliance on full episodes in MC:

- **From DP**: They take the concept of using the Bellman equations to iteratively update the value estimates but relax the requirement for a known model by using observed transitions instead.
- **From MC**: They learn directly from episodes, as MC methods do, but they update estimates based on individual steps rather than waiting for the episode to conclude, thus providing faster learning and greater flexibility in environments where episodes are very long or not well-Defined.

This unique combination makes TD methods particularly powerful in a wide array of environments, allowing for efficient, model-free learning that is both incremental and less dependent on the episode structure, thereby combining the best aspects of both DP and MC approach

c) What is meant by the terms on-policy and off policy in the context of TD methods? The discussion should include the equations for an on policy and off policy update. (4 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

Q4 Answer ALL parts. Total marks awarded for this question: 10 marks)

a) Compare and contrast the Symbolic AI and Machine Learning (ML) paradigms, and illustrate the answer with examples. (3 marks.)

Answer:

Symbolic AI and Machine Learning (ML) are two distinct paradigms within artificial intelligence (AI), each with its own approaches and methodologies for problem-solving.

Symbolic AI, also known as classical AI or good old-fashioned AI, relies on explicit rules and logical representations to simulate human reasoning. It involves the manipulation of symbols and rules to generate conclusions or actions based on input data. Examples of Symbolic AI include expert systems like MYCIN, which diagnose medical conditions based on a set of predefined rules, and automated reasoning systems such as chess engines that use predefined strategies to make decisions.

In contrast, **Machine Learning (ML)** involves systems that learn from data, identifying patterns without being explicitly programmed with domain rules. Instead of relying on hard-coded rules, ML algorithms adjust their parameters based on training data. For example, predictive analytics algorithms can predict consumer behavior based on past purchase data, while image recognition systems use convolutional neural networks (CNNs) to identify objects within images. ML excels in domains requiring pattern recognition and decision-making based on complex or voluminous data, offering adaptability and the ability to generalize to new situations based on the data it has been trained on.

b) Discuss four metric-based approaches used to quantify the performance of an ML algorithm, one example being precision and recall. (3 marks.)

Answer:

1. **1. Accuracy:**
    a. **Definition**: Accuracy measures the proportion of correctly classified instances out of the total number of instances.
    b. **Formula**:
    $Accuracy = TP + TN / TP + TN + FP + FN$
    c. **Example**: In a binary classification problem (e.g., spam detection), accuracy tells us the percentage of correctly classified emails out of all emails evaluated.

2. **Precision and Recall:**
    a. **Precision**:
        i. **Definition**: Precision measures the proportion of true positive (TP) instances among all positive predictions (TP + FP).
        ii. • **Formula**: $Precision = TP / TP + FP$

iii. • **Example**: In a medical diagnosis scenario, precision indicates the proportion of correctly identified patients with a specific condition out of all patients identified as having the condition.

b. **• Recall:**
   i. **Definition**: Recall measures the proportion of true positive (TP) instances that are correctly identified out of all actual positive instances (TP + FN).
   ii. **Formula**: $Recall = TP / TP + FN$
   iii. **Example**: In a search engine's relevance ranking, recall reflects the proportion of relevant documents retrieved out of all existing relevant documents.

3. **F1 Score:**
   a. **Definition**: The F1 score combines precision and recall into a single metric, providing a balanced measure of a classifier's performance.
   b. **Formula**: $F1\ Score = 2 \times Precision \times Recall / Precision + Recall$
   c. **Example**: The F1 score is commonly used in information retrieval and binary classification tasks, where both precision and recall are critical metrics for evaluating performance.

4. **4. Area Under the ROC Curve (AUC-ROC):**
   a. **Definition**: The AUC-ROC measures the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
   b. **Example**: In binary classification tasks (e.g., disease detection), the AUC-ROC assesses the classifier's ability to distinguish between positive and negative instances across different threshold settings, with a higher AUC indicating better performance. These metric-based approaches provide valuable insights into an ML algorithm's performance, enabling practitioners to evaluate and compare models effectively across different tasks and domains.

c) Describe Generative Adversarial Networks (GANs) OR Recurrent Neural Networks (RNNs). For the paradigm selected, discuss sample applications, training data, network structure, training, and constructs specific to the algorithm. For example, Long Short Term Memory (LSTM) in the case of RNNs. (4 marks.)

NOT COVERED

Q5 Answer ALL parts. Total marks awarded for this question: 10 marks.

a) List the steps in the REINFORCE Algorithm by Williams 1992. (3 marks.)

Answer: **ALREADY ANSWERED IN PREVIOUS PAPER**

b) Compare and contrast Q Learning with Policy Gradient (PG) approaches. Briefly describe two issues that can arise in PG approaches. (3 marks.)

Answer:

Learning vs. Policy Gradient (PG) Approaches

Q-Learning

1. **Definition**: Q-learning is a model-free, off-policy reinforcement learning algorithm that learns the optimal action-value function $Q(s,a)$Q(s,a) directly.
2. **Approach**: Q-learning learns the value of taking a specific action in a specific state by iteratively updating the Q-values based on the Bellman equation and the observed rewards.
3. **Advantages**:
    a. Simplicity: Q-learning is relatively straightforward to implement and understand.
    b. Stability: Q-learning tends to be more stable than policy gradient methods, particularly in environments with sparse rewards.

4. **Disadvantages**:
    a. **Exploration**: Q-learning can struggle with exploration in large state spaces or complex environments, potentially leading to suboptimal policies.
    b. **High Variance:** Q-learning can suffer from high variance in the estimates of action values, especially in environments with noisy or stochastic rewards.

Policy Gradient (PG) Approaches:

1. **Definition**: Policy gradient methods directly learn a parameterized policy $\pi\theta(a|s)$πθ(a|s) that defines the probability distribution over actions given states.
2. **Approach**: PG methods optimize the policy by estimating the gradient of the expected return with respect to the policy parameters and updating the parameters in the direction of this Gradient.
3. **Advantages:**
    a. **Exploration:** PG methods naturally encourage exploration by directly optimizing the policy's objective, potentially leading to better performance in complex environments.
    b. **Continuous Action Spaces:** PG methods can handle continuous action spaces more effectively than Q-learning, which typically requires discretization.
4. **Disadvantages:**
    a. **High Variance**: PG methods often suffer from high variance in gradient estimates, which can lead to slow convergence and instability during training.
    b. **Local Optima:** PG methods may converge to local optima rather than the globally optimal policy, particularly in high-dimensional or complex action spaces.

Two Issues in PG Approaches:

1. **High Variance in Gradient Estimates:** Policy gradient methods frequently encounter high variance in the estimates of the gradient of the expected return, especially when using Monte Carlo estimates or when dealing with long and sparse trajectories. This high variance can lead to slow convergence and unstable training dynamics.

2. **Local Optima:** PG methods are prone to converging to local optima rather than the globally optimal policy, particularly in high-dimensional or complex action spaces. This issue arises due to the non-convex nature of the policy optimization problem and the challenge of exploring the vast space of possible policies effectively.

c) The coding fragment is Figure Question-5 is an excerpt of an implementation of the REINFORCE method for the cartpole. Provide a detailed explanation for lines 7 and 8, and for the observation and action space in line 4. (4 marks.)

```
1. if __name__ == "__main__":
2.         env = gym.make("CartPole-v0")
3.         writer = SummaryWriter(comment="-cartpole-reinforce")
4.         net = PGN(env.observation_space.shape[0], env.action_space.n)
5.         print(net)
6.         agent = ptan.agent.PolicyAgent(net, preprocessor=ptan.agent.float32_preprocessor,
           apply_softmax=True)
7.         exp_source = ptan.experience.ExperienceSourceFirstLast(env, agent, gamma=GAMMA)
8.         optimizer = optim.Adam(net.parameters(), lr=LEARNING_RATE)
```

Figure Question-5. Adapted from Maxim Lapan. Deep Reinforcement Learning Hands-On. 2018.
Packt Publication

Answer: