**Solution to Exam Questions: Deep Reinforcement Learning (CS6482)**

**Q1: Deep Q Networks (DQN)**

**a) Experience Capture, Processing, and Sampling for Training a DQN (3 marks)**

**Experience Capture:**

* DQNs use **experience replay** to store experiences as tuples:

bash

CopyEdit

(state, action, reward, next\_state, done)

* Stored in a **replay buffer**.

**Experience Processing:**

* The buffer holds a **fixed number of recent experiences**.
* Older experiences are discarded as new ones come in (FIFO).

**Sampling for Training:**

* Randomly sample **mini-batches** from the replay buffer to break correlation between sequential observations.
* Feed these batches to the **neural network** for training.

**Coding Fragment:**

python

CopyEdit

replay\_buffer = []

for episode in range(max\_episodes):

replay\_buffer.append((state, action, reward, next\_state, done))

if len(replay\_buffer) > batch\_size:

minibatch = random.sample(replay\_buffer, batch\_size)

**b) Target Used to Train a DQN (3 marks)**  
The **target** in DQN is the **Q-value** predicted using the next state:

r

CopyEdit

target = reward + gamma \* max(Q(next\_state, a'))

* Uses the **Bellman Equation** for updating:

css

CopyEdit

Q(state, action) ← Q(state, action) + α [target - Q(state, action)]

* **Coding Fragment:**

python

CopyEdit

target = reward + gamma \* np.max(target\_model.predict(next\_state))

loss = mse(current\_q\_value, target)

**c) Cause of Maximization Bias and Reduction (4 marks)**  
**Cause:**

* DQNs tend to **overestimate Q-values** because they use the same model to select and evaluate the action.

**Reduction Approach:**

* **Double DQN:** Uses separate networks to select and evaluate the best action.

ini

CopyEdit

target = reward + gamma \* Q(next\_state, argmax(Q(next\_state, a; θ); θ−))

**Q2: CNN Architectures and Batch Normalization**

**a) Parameters in GoogleLeNet vs. AlexNet (3 marks)**

* **AlexNet:** 60 million parameters due to large **fully connected layers**.
* **GoogleLeNet (Inception):** 6 million parameters due to **Inception modules**, which use **1x1 convolutions** for dimensionality reduction.

**Calculation:**

* **Inception Modules:** Combine **1x1, 3x3, and 5x5 filters** efficiently, reducing the parameter count compared to **stacked convolutions** in AlexNet.

**b) Batch Normalization Layers (3 marks)**

* **Number of Parameters:** Each **Batch Normalization layer** has two trainable parameters: **γ (scale)** and **β (shift)**.
* **Calculation:**
  + 3 BatchNorm layers x 2 parameters each = **6 trainable parameters**
* **Code Fragment:**

python

CopyEdit

model.add(keras.layers.BatchNormalization())

**c) Key Concept in ResNet (4 marks)**

* **Residual Learning:**
  + Introduces **skip connections** to bypass one or more layers.

ini

CopyEdit

y = F(x, {Wi}) + x

* **1x1 Convolution (Stride 2):**
  + Reduces **dimension** while maintaining **shortcut connections**.
  + Efficient for **downsampling** without losing information.

**Q3: Policy Gradient Methods**

**a) Steps in REINFORCE Algorithm (3 marks)**

1. Collect **trajectories** using the current policy.
2. Compute **returns (Gt)** for each step.
3. Update policy using the gradient:

scss

CopyEdit

θ ← θ + α \* ∇L(θ)

**b) Three Advantages of Policy Gradient Approaches (3 marks)**

1. **Direct Optimization:** Directly learns the **policy** rather than estimating Q-values.
2. **Stochastic Policies:** Suitable for environments requiring **random actions**.
3. **Continuous Action Spaces:** Efficient for tasks like **robotic control**.

**c) Explanation of Code Fragment (4 marks)**

* **Lines 2-11:** Implement the **REINFORCE algorithm** to optimize the policy.
* **Line 3:** Computes **total rewards** for multiple episodes.
* **Line 4:** Uses **discounting** to calculate future returns.
* **Line 5-11:** Averages gradients over episodes and updates the model.
* **Optimizer:** Uses **Gradient Descent** to adjust weights.

**Q4: Learning Paradigms and Symbolic Systems**

**a) Physical Symbol System Hypothesis (3 marks)**

* Claims that **intelligent behavior** arises from symbol manipulation.
* **Comparison with ML:**
  + PSSH: Uses **symbolic reasoning**.
  + ML: Uses **pattern recognition and learning**.

**b) Evolutionary Computation vs. Reinforcement Learning (3 marks)**

| **Aspect** | **Evolutionary Computation** | **Reinforcement Learning** |
| --- | --- | --- |
| Optimization Approach | Population-based (genetic algorithms) | Learning from interactions (rewards) |
| Adaptation Mechanism | Mutation and crossover | Policy improvement and value iteration |
| Use Case | Genetic programming, optimization | Game playing, robotics |

**c) Learning Paradigm: LSTM (4 marks)**

* **Training Algorithm:** Uses **Backpropagation Through Time (BPTT)**.
* **Network Structure:** Consists of **input, forget, and output gates**.
* **Training Data:** Sequential data (e.g., text or time series).
* **Applications:**
  + **Text Generation**
  + **Speech Recognition**
  + **Sequence Prediction**

**Q5: Reinforcement Learning Concepts**

**a) Definition and Key Issues (3 marks)**

* **Reinforcement Learning (RL):** Learning through **trial and error** to maximize rewards.
* **Key Issues:**
  + **Exploration vs. Exploitation:** Balancing new actions with known rewards.
  + **Credit Assignment:** Identifying which actions lead to rewards.
  + **Delayed Reward:** Evaluating long-term gains from actions.

**b) Sarsa vs. Q-Learning Path (3 marks)**

* **Plot:** Sarsa computes a **safer path** by choosing actions based on the **current policy**, while Q-learning chooses the **optimal action** without considering risk.
* **Reason:** Sarsa is **on-policy** (updates based on actions actually taken), while Q-learning is **off-policy** (updates based on the optimal action).

**c) On-Policy vs. Off-Policy in TD Methods (4 marks)**

* **On-Policy:** Learns from the **policy it is currently using**. (e.g., **Sarsa**)

css

CopyEdit

Q(s, a) ← Q(s, a) + α [r + γ Q(s', a') - Q(s, a)]

* **Off-Policy:** Learns from the **optimal policy**, irrespective of current actions. (e.g., **Q-Learning**)

css

CopyEdit

Q(s, a) ← Q(s, a) + α [r + γ max Q(s', a') - Q(s, a)]

* **Difference:**
  + On-Policy: Learns from **actual experience**.
  + Off-Policy: Learns from **optimal actions**, even if not executed.