The City College of New York

CSC 59866-E: Senior Project I

AI Agents for Decision Making in the Real World

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu, sbandyopadhyay@gc.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York

February 23, 2026 CSC 59866

Deep Learning, Deep Reinforcement Learning (RL) & Multi-Agent Deep Reinforcement Learning (MARL): Training/Finetuning AI Models, Transformers, Q Learning, Deep Q Networks and the Bellman Equation

Logistics

Recent Milestone: Coding Assignment 1 is released!

- Due March 8, 2025 11:59 PM EST
- Take a look at the assignment early on so that you can ask me questions e.g. about using Google Colab, How to Use a GPU runtime, etc.

Next Goal: We will be finalizing Project Groups and Project Topics this week!

Our AI Agents Journey So Far

Lecture 4: We learned Tabular Q-Learning and the Bellman Equation.

Lecture 5: We discussed Reward Modeling and how agents learn to "hack" bad rewards.

Lecture 6 & 7: We covered Distributed Processing, Modalities, and physical network constraints (Latency vs. Bandwidth).

Today: We unite these concepts. How do we scale the Bellman Equation using Deep Learning to solve real-world complexities?

Today's Agenda

Deep Learning & Transformers

- Self-Attention, Encoders, and Decoders

Deep Reinforcement Learning

- Deep Q Networks

Multi-Agent Deep Reinforcement Learning

- Centralized Training Decentralized Execution

Deep Learning & Transformers

Deep Learning Refresher

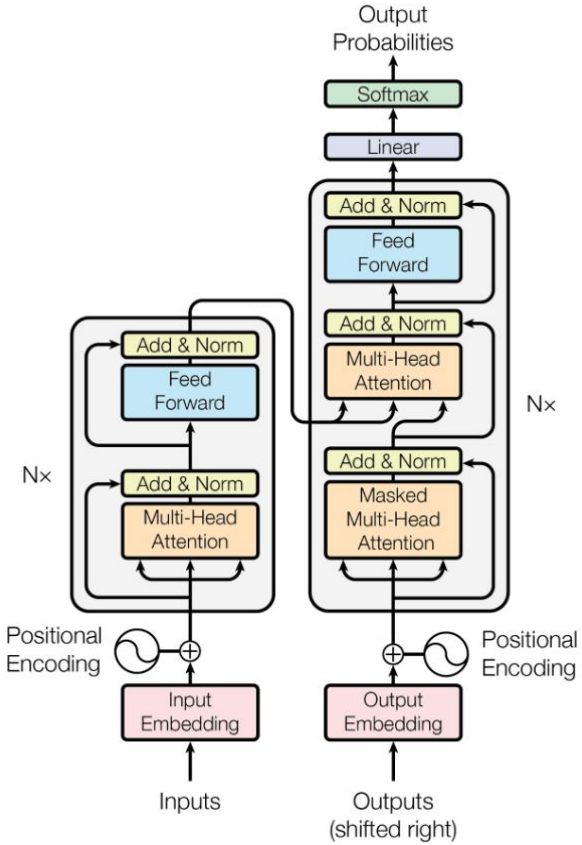
- **The Concept:** Instead of memorizing explicit rules or tables, Deep Learning uses multi-layered artificial neural networks to learn representations of data.
- **Why "Deep"?** Multiple hidden layers allow the model to learn hierarchical features (e.g., Layer 1 learns edges, Layer 2 learns shapes, Layer 3 learns a car).
- **Inference:** A forward pass through the network (computationally fast, great for Edge deployment!).
- **Training:** Backpropagation and Gradient Descent (computationally heavy).

Transformers

Before 2017, Recurrent Neural Networks (RNNs) read data sequentially. Transformers read everything at once.

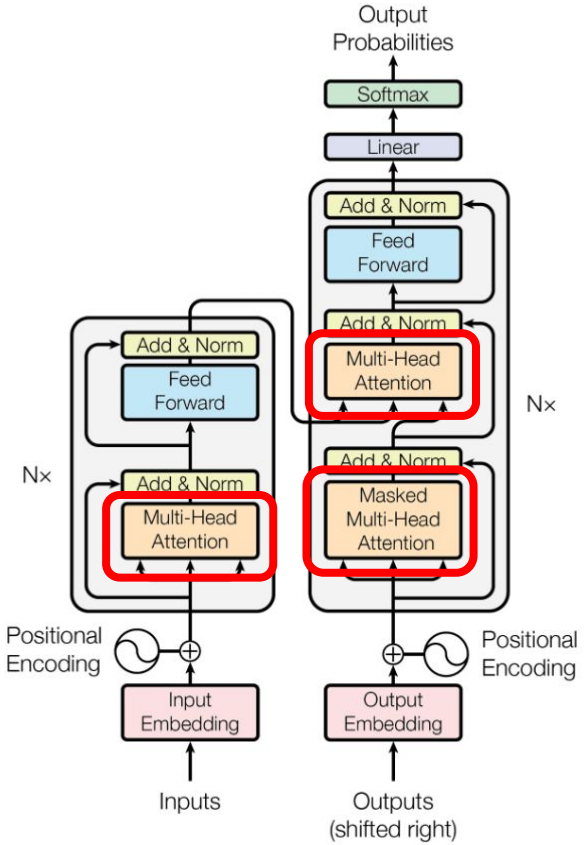
Self-Attention: The mechanism that allows the model to weigh the importance of every part of the input data simultaneously.

Relevance to AI Agents: Transformers aren't just for text (LLMs). They are now the backbone of Multi-Modal Agents (Vision-Language Models) and are actively used to process sequences of actions/states in RL (Decision Transformers).



Self-Attention

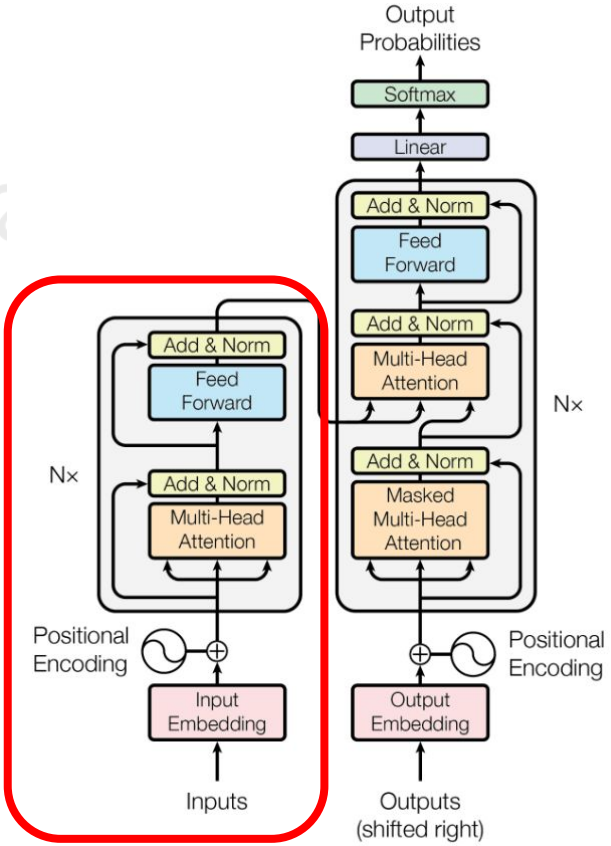
- The fundamental innovation is the **Multi-Head Attention** block.
- Instead of processing a sequence one step at a time, it computes how relevant *every* element in the sequence is to *every other* element simultaneously.
- *In the Diagram:* This is the orange block that allows the network to "attend" to different parts of its own input before moving on.



The Encoder (Left Block)

Reads input data and build a rich, contextual representation.

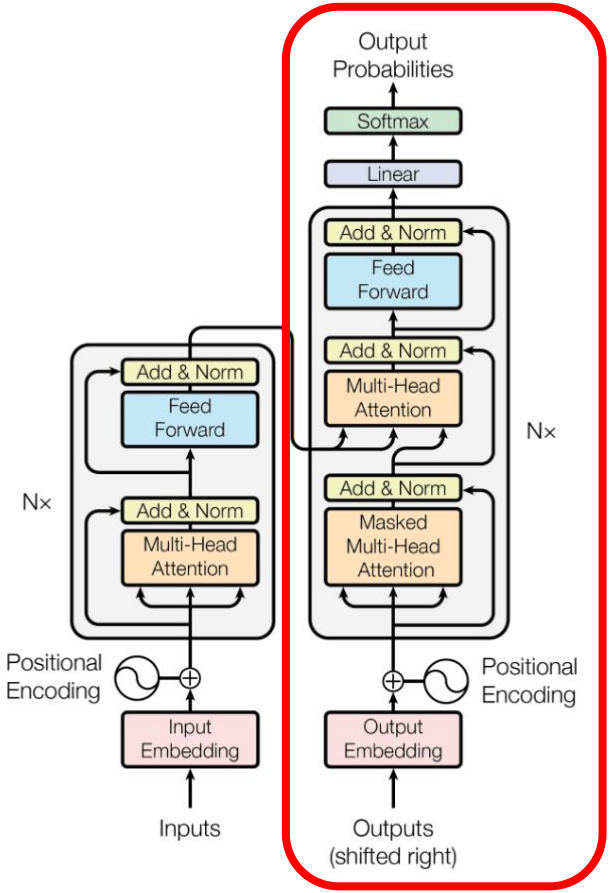
- **Input Embedding & Positional Encoding:** Turns raw data (e.g., words or pixels) into vectors and injects information about order (since the model isn't sequential).
- **Attention Block:** Allows the model to weigh the relationship between different inputs.
- **Feed Forward Block:** A standard neural network layer that processes the output of the attention block.
- **Normalization & Residual Connections (Add & Norm):** Crucial for training deep networks stably.



The Decoder (Right Half)

Goal: Generate the final output based on the Encoder's understanding.

Key Difference: It has **Masked** Attention. When generating the *next* item in a sequence, it can't be allowed to "peek ahead" at the future. It can only attend to past outputs and the full input from the encoder.



Training vs. Fine-Tuning

- **Pre-Training (The Foundation):** Training a massive model from scratch on huge datasets (e.g., the entire internet). Costs millions of dollars.
- **Fine-Tuning (The Adaptation):** Taking a pre-trained model and updating its weights on a small, specific dataset.
    - *Example:* Fine-tuning a generic LLM on medical textbooks so it becomes a surgical assistant.
- **Project Tip:** *Do not pre-train models from scratch in this class.* Use open-source foundation models (via HuggingFace) and fine-tune them for your specific tasks or use techniques at inference (e.g. Multi-Agent Debate).

Deep Reinforcement Learning

Recap: The Bellman Equation & The Curse of Dimensionality

The Bellman Equation: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

The Tabular Limit: In Lecture 4, we saw that a Q-Table requires an entry for every possible State-Action pair.

The Real-World Problem: If your state is a 256×256 RGB camera image, the number of possible states is $256^{256 \times 256 \times 3}$. A table cannot hold this. We call this the **Curse of Dimensionality**.

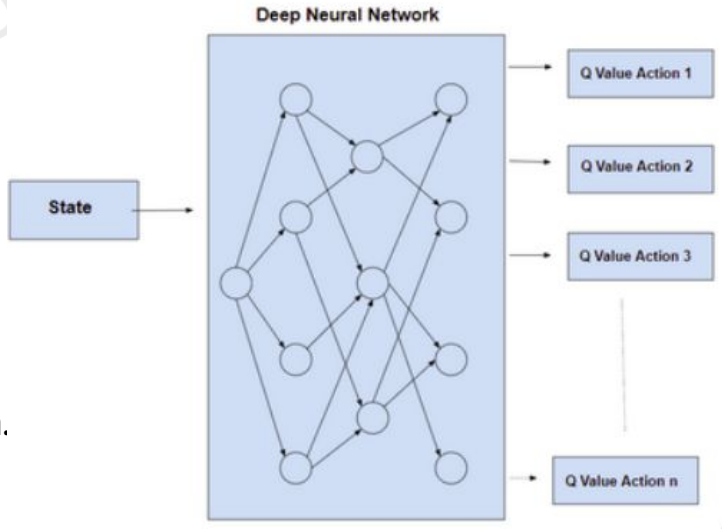
Deep Q Learning

Replace the Q-Table with a Deep Neural Network.

How it works:

- **Input:** The current state (e.g., pixels from a camera, or vector of sensor readings).
- **Network:** Processes the state to extract features.
- **Output:** A predicted Q-Value for *every* possible action.

The Breakthrough: This was the architecture DeepMind used to beat humans at Atari using raw pixels as input.



Deep Q-Learning Updates

Instead of updating a specific cell in a table, we update the *weights* of the neural network.

Target Value: $y = r + \gamma \max_{a'} Q(s', a'; \theta_{target})$

Loss Function: We minimize the Mean Squared Error (MSE) between the predicted Q-value and the Target Value (we use a *target* network here to avoid training instability).

Experience Replay: The agent stores past memories (state, action, reward, next_state) in a buffer and samples them randomly to train the network, breaking harmful correlations in sequential data.

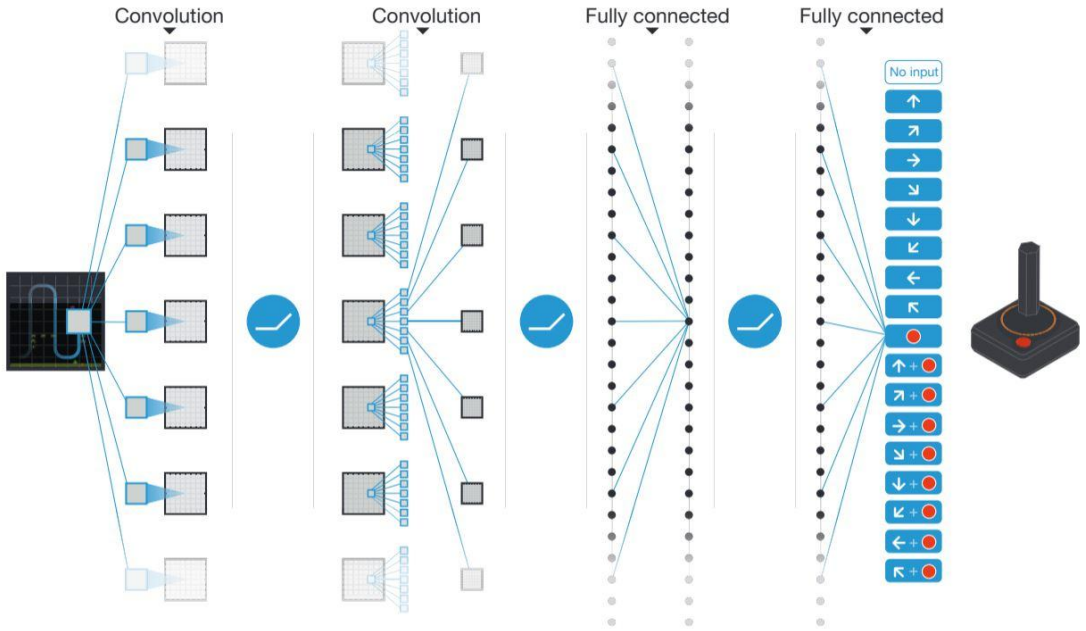
Example: Atari Games

Input: The state (last four game frames)

Network:

- Convolutional layers (useful for image data)
- Fully connected layers (normal neural network)

Output: A vector of q-values, one for each action (controller input)



Multi-Agent Deep Reinforcement Learning

Scaling to MARL

The Jump: What happens when we put multiple DQNs in the same environment?

The Non-Stationarity Problem (Revisited): From the perspective of Agent 1, Agent 2 is just part of the environment. If Agent 2's neural network updates and changes its behavior, Agent 1's environment just unexpectedly shifted.

The Consequence: Independent Deep Q-Learning (each agent learns their own DQNs without considering other agents) in multi-agent settings is notoriously unstable. The agents chase a moving target and often fail to converge.

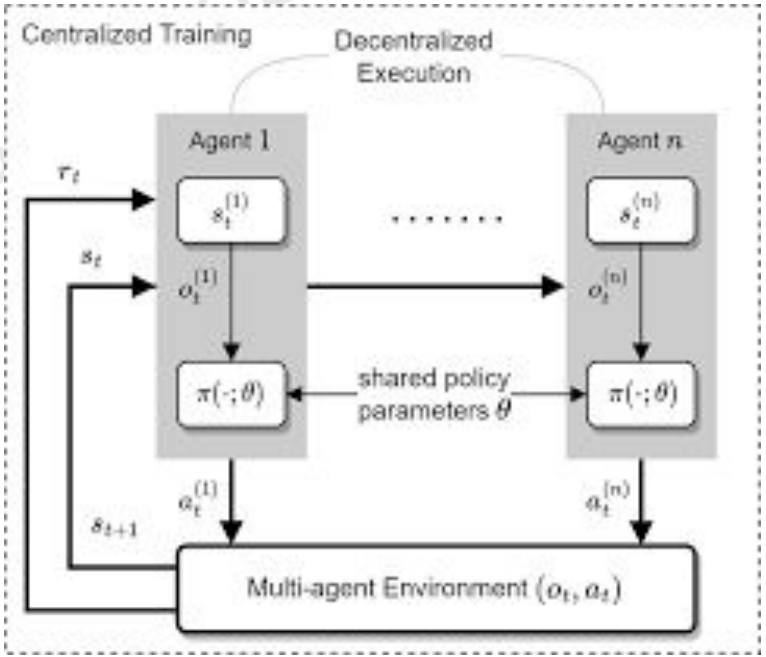
Centralized Training with Decentralized Execution

- **The Standard MARL Architecture:** The dominant paradigm for solving the non-stationarity problem in complex multi-agent environments.
- **Centralized Training (The Global View):**
  - During training in the simulator, we use a central "Critic" neural network.
  - The Critic gets to see the *Global State* (everything) and the actions of all agents. It evaluates how good the team's joint action was.
- **Decentralized Execution (The Edge):**
  - During deployment (the real world), the Central Critic is thrown away.
  - The individual agents ("Actors") only use their own local observations.
- **Relevance:** This is how most complex coordinating agents (like self-driving car fleets or drone swarms) are trained today.

Unpacking CTDE

The outer dotted boundary illustrates Centralized Training, meaning the system has full access to the global state ( ) and global rewards ( ). s_t r_t

The solid grey boxes represent Decentralized Execution, where individual agents operate independently.

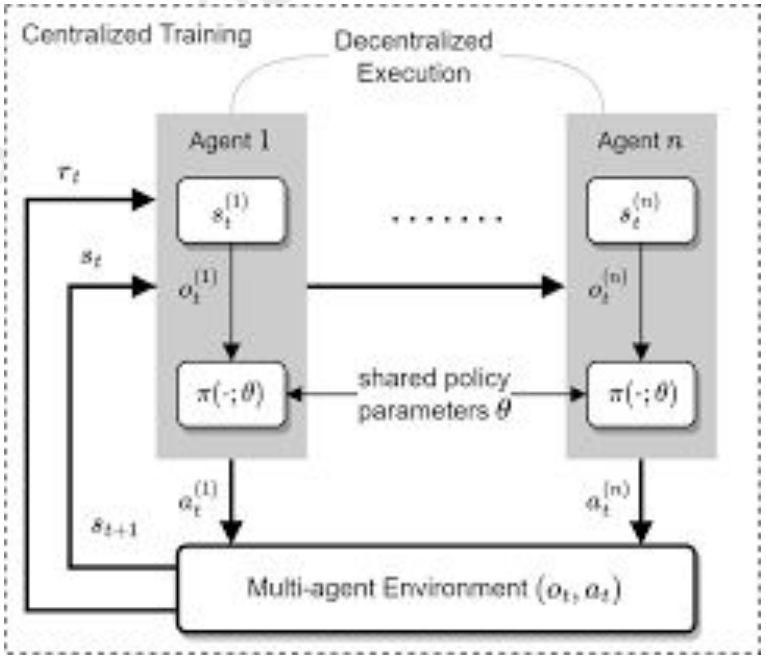


Unpacking CTDE (Cont.)

During execution, each Agent n relies on its local observation ($o_t^{(n)}$) rather than the global state.

The agents use shared policy parameters (θ) within their policy function (π) to map observations to actions ($a_t^{(1)}$, $a_t^{(n)}$).

These individual actions are fed into the Multi-agent Environment, which then advances to the next state (s_{t+1}).



Summary & Next Steps

- **Deep Learning** allows us to approximate the world rather than memorizing it.
- **DQNs** map complex, high-dimensional states to optimal actions using the Bellman Equation as a loss function.
- **CTDE** is the architectural trick we use to train cooperative multi-agent teams without requiring them to share infinite bandwidth in the real world.
- **Next Class:** Exploring Open-Source Tools and simulation environments (JaxMARL, Gymnasium, PettingZoo) to actually build these networks.

Questions?