

GPT-2 with Q-Head: Technical Overview

1. Abstract

This document describes a method for augmenting GPT-2 with a Q-head that predicts expected future returns for each possible next token. The approach combines standard language modeling with reinforcement learning concepts, enabling the model to learn both token prediction and value estimation simultaneously.

2. Main Idea

The core insight is to treat autoregressive language generation as a sequential decision-making process. At each position t in a sequence, the model must choose the next token a from the vocabulary V . We augment GPT-2 with a second output head that predicts $Q(s_t, a)$ — the expected discounted future return if action a is taken in state s_t .

Architecture Overview:

- **Backbone:** GPT-2 transformer producing hidden states h_t of dimension H
- **LM Head:** Linear projection $h_t \rightarrow$ logits over vocabulary (standard next-token prediction)
- **Q Head:** Linear projection $h_t \rightarrow$ Q-values over vocabulary (value prediction for each action)

The Q-head outputs a $|V|$ -dimensional vector at each position, where each component $Q(s_t, a)$ represents the expected future return if token a is selected as the next token. This allows querying the value of any action without needing to actually sample it.

3. Objective Function

The training objective combines two loss terms:

$$L_{\text{total}} = L_{\text{LM}} + \lambda \cdot L_Q$$

where λ (q_weight) controls the relative importance of the Q-learning objective.

3.1 Language Modeling Loss (L_{LM})

Standard causal language modeling loss using cross-entropy. For a sequence of tokens (x_1, x_2, \dots, x_T) , the model predicts each token given all previous tokens:

$$L_{\text{LM}} = -\sum_{t=1}^{T-1} \log P(x_{t+1} | x_1, \dots, x_t)$$

This is computed using shifted logits: predictions at positions 0.. $T-2$ are compared against ground truth tokens at positions 1.. $T-1$. Padding positions are masked out.

3.2 Q-Value Loss (L_Q)

The Q-head is trained using mean squared error between predicted Q-values and discounted return targets. For the observed action a_t (the actual next token), we supervise:

$$L_Q = (1/N) \cdot \sum_t (Q(s_t, a_t) - G_t)^2$$

where G_t is the discounted return from position $t+1$ onward:

$$G_t = \sum_{k=t+1}^{T-1} \gamma^{k-(t+1)} \cdot r_k$$

Here γ is the discount factor (default 0.99) and r_k is the reward at position k . Note that G_t represents future rewards starting from the NEXT position, not the current one.

4. Computing Discounted Returns

The discounted returns are computed via dynamic programming, working backwards through the sequence:

```
def compute_discounted_returns(rewards, gamma):
    returns = zeros_like(rewards)
    future = 0.0
    for t in range(T-1, -1, -1):
        returns[t] = future
        future = rewards[t] + gamma * future
    return returns
```

This ensures that $returns[t]$ contains the discounted sum of all rewards from $t+1$ to $T-1$, which is exactly what $Q(s_t, a_t)$ should predict for the optimal policy.

5. Key Properties

- 1. On-Policy Supervision:** The Q-head is only supervised for the actually observed next token (the action that was taken). This is a form of on-policy learning where we learn Q-values along the trajectory defined by the training data.
- 2. Vocabulary-Sized Q-Output:** Unlike scalar value functions $V(s)$, the Q-head outputs a value for every possible action. This enables action selection via $\text{argmax}_a Q(s, a)$ at inference time.
- 3. Shared Representations:** Both heads share the transformer backbone, allowing the Q-head to benefit from the rich representations learned for language modeling.
- 4. Flexible Reward Specification:** Rewards can encode any per-token signal — correctness, style, safety scores, or learned reward models. The current implementation uses synthetic random rewards for demonstration.

6. Extensions and Future Directions

The current implementation provides on-policy Q-learning. For more sophisticated applications, consider:

- **Off-policy learning:** Using importance sampling or model-based rollouts to learn Q-values for actions not taken in the training data.
- **Temporal difference learning:** Using TD(0) or TD(λ) updates instead of Monte Carlo returns.
- **Actor-critic methods:** Using the Q-head to guide policy updates for the LM head.
- **Reward modeling:** Training a separate reward model and using its outputs as r_t .

7. Summary

The GPT2-with-Q-head architecture demonstrates how reinforcement learning concepts can be integrated into language models. The combined objective:

$$L = L_{LM} + \lambda \cdot L_Q$$

trains the model to simultaneously predict next tokens (via cross-entropy) and estimate expected future returns (via MSE on discounted returns). This provides a foundation for value-guided text generation and reinforcement learning from human or automated feedback.