Learning to Learn: Curriculum Selection via Soft Q-Learning for Sample-Efficient Language Models

Scott Viteri Damir Vrabac

1 Overview

We propose an algorithm that enables language models to learn what data to learn from. Rather than training on randomly sampled data, the model takes actions that select training examples, receiving reward based on how well it currently predicts held-out data. A Q-function, parameterized as a small network reading the language model's internal representations, learns to predict the long-run value of each candidate. The policy is derived from the Q-function via a Boltzmann distribution, eliminating the need for a separate policy network. The core hypothesis is that this learned curriculum selection will yield significantly better sample efficiency than random or heuristic curricula.

2 Technical Approach

2.1 Setup

- Language model (GPT-2) with parameters θ, hidden dimension $d = 768$
- Streaming data source, yielding fresh batches of N candidate context windows each step
- Fixed held-out evaluation set D, large; a fresh random subset $\hat{D} \subset D$ of size M is sampled each step for reward computation
- Q-network Q_ϕ with parameters ϕ, implemented as a 2-layer MLP ($d \to 32 \to 1$) reading the base model's residual stream (see Section 2.3)
- Target Q-network $Q_{\bar{\phi}}$ with parameters $\bar{\phi}$, a Polyak-averaged copy of Q_ϕ used for stable bootstrap targets
- Boltzmann temperature $\beta > 0$ controlling exploration
- Learning rate α for the LM gradient step; learning rate η for the Q-network update
- Discount factor $\gamma \in (0, 1)$ controlling the effective planning horizon
- Polyak averaging rate $\tau \in (0, 1)$ for the target network update
- Gradient clipping threshold G (max global norm; applied to both LM and Q-network gradient steps)

2.2 State, Action, and Reward

The state s_k is the current model parameters θ_k, which determine the model's representations and therefore the features available to the Q-network. In practice, s_k is never represented explicitly—it is accessed implicitly through the base model's hidden states when candidates are forwarded through it (Section 2.3).

The action $a_k \in C_k$ is the choice of which training example to train on at step k, where C_k is the batch of N candidates available at that step. Since data is streamed, each step presents a

fresh candidate batch, and each candidate is seen at most once. The Q-network generalises across batches via the base model's hidden-state representation.

The reward is the held-out log-probability *after* the training update:

$$r_k = \frac{1}{M} \log p_{\theta_{k+1}}(\hat{D}_k), \tag{1}$$

where $\hat{D}_k$ is a fresh random subset of D of size M, and θ_{k+1} results from training on a_k. This measures the model's absolute predictive fitness as a consequence of the selected action. Resampling $\hat{D}_k$ each step prevents the Q-function from overfitting to a fixed evaluation subset.

The Q-function optimises the discounted return $(1-\gamma)\sum_k \gamma^k r_k$. The weights $(1-\gamma)\gamma^k$ sum to one, so this is the *exponentially weighted average* of the reward sequence: the Q-value at any state equals the typical per-step reward expected over the next $\sim 1/(1-\gamma)$ steps under the current policy. Maximising this objective favours curricula that achieve high predictive fitness quickly—each step spent with poor predictions is penalised, with a horizon controlled by γ.

2.3 Q-Network Architecture

The Q-network reads the base model's internal representations to assess candidate value. For each candidate $x \in C_k$:

1. Forward x through the base model with parameters θ_k, extracting the last-layer hidden state at the final token position: $h_x = \texttt{last\_hidden}(\theta_k, x) \in \mathbb{R}^d$.
2. Detach h_x from the computation graph (no gradients flow into θ).
3. Pass through the Q-network:

$$Q_\phi(s_k, x) = W_2\, \sigma(W_1\, h_x + b_1) + b_2, \tag{2}$$

   where $\phi = \{W_1 \in \mathbb{R}^{32\times d},\ b_1 \in \mathbb{R}^{32},\ W_2 \in \mathbb{R}^{1\times 32},\ b_2 \in \mathbb{R}\}$ and σ is ReLU.

The hidden state h_x implicitly encodes both the model's current state (through θ_k, which determines the representations) and the identity of the candidate (through the input x). By detaching h_x, we ensure that θ is updated only by the language modeling objective, while ϕ is updated only by the TD loss. This prevents the co-adaptation instability that arises when value gradients distort the base model's representations.

2.4 Normalised Discounted Soft Bellman Equation

The Q-function satisfies a normalised discounted soft Bellman equation:

$$Q(s_k, a_k) = (1-\gamma)\, r_k + \gamma\, V_{\bar{\phi}}(s_{k+1}), \tag{3}$$

where $\gamma \in (0,1)$ is the discount factor and $V_{\bar{\phi}}$ is the soft value computed from the target network (see below). The $(1-\gamma)$ factor on the reward is a normalisation that keeps Q-values on the same scale as the reward regardless of γ: a constant reward r gives $Q^* = r + \gamma\,\beta \log N/(1-\gamma)$ at the fixed point, where the first term is $O(r)$ rather than $O(r/(1-\gamma))$. This prevents Q-value inflation and keeps the MLP output in a numerically comfortable range.

The soft value function is:

$$V(s) = \beta \log \sum_{a'} \exp\big(Q(s,a')/\beta\big), \tag{4}$$

so the Bellman equation reads $Q(s_k, a_k) = (1-\gamma)\, r_k + \gamma\, V_{\bar{\phi}}(s_{k+1})$. At convergence, $Q(s, a)$ represents a normalised discounted sum of future rewards from choosing a in state s and following the Boltzmann policy thereafter. With $\gamma = 0.9$, the effective horizon is $\sim$10 steps.

The normalisation bounds the reward contribution to Q-values: since $|r_k| \leq R_{\max}$, the reward term contributes at most $(1-\gamma)R_{\max}/(1-\gamma) = R_{\max}$ to the Q-value fixed point. The entropy bonus from the soft value adds $\gamma\,\beta \log N/(1-\gamma)$, which is a constant offset shared by all actions and does not affect the policy ranking.

Target network. To stabilise the bootstrap target $V(s_{k+1})$, we maintain a target network $Q_{\bar{\phi}}$ that is a Polyak-averaged copy of the online network Q_ϕ:

$$\bar{\phi} \;\leftarrow\; \tau\,\phi + (1-\tau)\,\bar{\phi}, \qquad \tau \in (0, 1). \tag{5}$$

The soft value in the TD target is computed from $Q_{\bar{\phi}}$, not Q_ϕ. This breaks the feedback loop where the network's own output serves as its training target, preventing the oscillations and divergence that can occur with function approximation. The online network Q_ϕ is still used for action selection (the Boltzmann policy) and for the prediction $\hat{Q}$ in the TD loss. With $\tau = 0.01$, the target network tracks the online network with a lag of $\sim 1/\tau = 100$ steps.

2.5 Algorithm

The key implementation insight is that each step's forward pass of candidates through the base model serves two purposes: action selection for the *current* step and the bootstrap target for the *previous* step's Q update. This avoids a redundant second forward pass.

Computational cost per step. One forward pass of N candidates through the base model (batched, no gradient); one forward-backward pass of the selected example for LM training; one forward pass of M held-out examples for reward; one MLP forward-backward for the Q update; one MLP forward pass through the target network (negligible). The candidate forward pass dominates; the MLP and reward computations are comparatively cheap.

Staleness. The Q update at step k uses the target network $Q_{\bar{\phi}}$ for the bootstrap value V_k, and the online network Q_ϕ for action selection. The target network lags the online network by $\sim 1/\tau$ steps, which is by design: this lag stabilises the TD target. The online Q-values q_k used for action selection have one-step staleness (computed before the ϕ update), which is standard in online Q-learning and negligible given the small MLP learning rate η.

2.6 Soft Mixture Training Variant

In the base algorithm (Section 2.5), the Boltzmann policy samples a single action $a_k \in C_k$ and the LM trains on that one example. This is sample-inefficient in a specific sense: at every step, N candidates are already forwarded through the base model for Q-scoring, but only one of them contributes to the LM gradient update.

Expected gradient under the policy. An alternative is to train the LM on a *policy-weighted mixture* of all N candidates. Define the Boltzmann weights:

$$\pi_i = \frac{\exp(Q_\phi(s_k, x_i)/\beta)}{\sum_{j=1}^{N} \exp(Q_\phi(s_k, x_j)/\beta)}, \qquad x_i \in C_k. \tag{6}$$

The LM loss becomes:

$$\mathcal{L}_{\text{mix}}(\theta; C_k) = \sum_{i=1}^{N} \pi_i \, \ell(\theta, x_i), \tag{7}$$

where $\ell(\theta, x_i)$ is the per-token cross-entropy loss on candidate x_i. The LM gradient step is then:

$$\theta_{k+1} = \theta_k - \alpha \, \nabla_\theta \mathcal{L}_{\text{mix}}(\theta_k; C_k). \tag{8}$$

This is precisely the *expected gradient* under the Boltzmann policy:

$$\nabla_\theta \mathcal{L}_{\text{mix}} = \mathbb{E}_{x \sim \pi(\cdot|s_k)}\big[\nabla_\theta \ell(\theta_k, x)\big]. \tag{9}$$

In the base algorithm, the LM gradient is a single-sample Monte Carlo estimate of this expectation. The soft mixture computes it exactly, eliminating the variance from action sampling.

Q-learning is unchanged. The Q-network update proceeds exactly as in Section 2.5: a discrete action a_k is still sampled from the Boltzmann policy, its hidden state h_{prev} is stored, and the standard TD update is applied at the next step. The Q-function continues to satisfy the discounted soft Bellman equation (3). The only change is in how the environment *transitions*: the LM parameters are updated via the full mixture rather than a single example.

Off-policy interpretation. From the Q-learner's perspective, the situation is analogous to off-policy learning. The Q-learner observes: "I selected action a_k, the environment transitioned to state s_{k+1}, and I received reward r_k." It does not need to know that the transition was caused by a mixture-weighted gradient step rather than a single-example step. Q-learning is inherently off-policy—the Bellman backup $Q(s, a) = (1 - \gamma)\, r + \gamma \, V(s')$ is valid regardless of the behavior policy that generated the transition, as long as r and s' are the actual observed reward and next state.

In this case, the "behavior policy" (the mixture-weighted LM update) is a smooth, deterministic function of the same Boltzmann policy that the Q-learner is evaluating. Over many steps, the Q-function converges to predict the value of upweighting a given candidate within the mixture—which is exactly the marginal contribution signal needed to set the mixture weights well.

Computational cost. The N candidate forward passes for Q-scoring are already performed (under `torch.no_grad`). The soft mixture additionally requires N forward-backward passes through the full model for loss computation, processed in mini-batches with gradient accumulation. This replaces the single forward-backward pass of the base algorithm. The cost increase is a factor of $\sim N$ in the LM training portion of each step, but the LM training step was already cheap relative to the N candidate forward passes and M held-out forward passes. The net effect is roughly a $2\times$ increase in total step time (the candidate extraction forward passes, which dominate, are shared).

When to use soft mixture training. The soft mixture variant is most beneficial when:

- The number of candidates N is moderate (the gradient accumulation cost scales linearly in N).
- Variance reduction matters: early in training when Q-values are noisy, the expected gradient is more stable than a single-sample estimate.
- The effective batch size increase is desirable: the mixture-weighted update is equivalent to training on a soft batch of $\sim 1/\|\pi\|_2^2$ effective examples per step (the inverse of the collision probability of the policy distribution).

The flag `--soft_mixture` enables this variant. When disabled (the default), the algorithm reverts to single-action selection as in Algorithm 1.

2.7 Interpretation as Free-Energy Minimisation

The Boltzmann policy is the solution to a free-energy minimisation problem at each state:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \Big[-\mathbb{E}_{x\sim\pi}\big[Q_\phi(s,x)\big] + \beta\,\mathrm{KL}\big(\pi \,\|\, p_0\big) \Big], \tag{10}$$

where p_0 is the uniform distribution over C_k. Since p_0 is uniform, $\mathrm{KL}(\pi\|p_0) = \log N - H(\pi)$, and the objective reduces (up to constants) to:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \Big[-\mathbb{E}_{x\sim\pi}\big[Q_\phi(s,x)\big] - \beta\,H(\pi) \Big]. \tag{11}$$

The first term encourages selecting high-value candidates; the second encourages exploration. The temperature β controls the tradeoff.

ELBO interpretation. Define a latent variable model where the "evidence" is the event that the current trajectory is optimal, with likelihood $p(\text{optimal} \mid x) \propto \exp(Q(s,x)/\beta)$. The policy π acts as a variational posterior over actions, and the free energy is the negative ELBO:

$$\log p(\text{optimal} \mid s) \;\geq\; \mathbb{E}_{x\sim\pi}\left[\frac{Q(s,x)}{\beta}\right] - \mathrm{KL}(\pi \,\|\, p_0). \tag{12}$$

The Boltzmann policy makes this bound tight. The soft value $V(s) = \beta \log \sum_x \exp(Q(s,x)/\beta)$—which appears as the bootstrap term in the TD target—is the log-evidence, measuring the total accessible value in the current candidate batch.

3 Evaluation

3.1 Primary Metric: Sample Efficiency

The central question is: how many training examples does the model need to reach a given performance level?

We will measure:

- Perplexity on a fixed evaluation set as a function of training examples seen
- Performance on downstream tasks (e.g., MMLU, HellaSwag) as a function of training examples

3.2 Baselines

- **Random curriculum:** Uniform sampling from candidates
- **Loss-based curriculum:** Prioritise high-loss examples
- **Uncertainty-based curriculum:** Prioritise examples with high model uncertainty
- **Competence-based curriculum:** Examples ordered by difficulty (requires difficulty labels)

3.3 Analysis

Beyond aggregate metrics, we will examine:

- **Curriculum structure:** Does the learned curriculum exhibit interpretable patterns? Developmental stages? Topic clustering?
- **Exploration dynamics:** How does the policy entropy evolve over training? Does the temperature β produce reasonable exploration?
- **Q-function interpretability:** Which candidates receive high Q-values at different stages of training? Does the Q-function learn to identify examples that are valuable given the model's current state?
- **Q-value dynamics:** How do Q-values evolve over training? Convergence of Q-values indicates the value function has stabilised; the spread (standard deviation) reflects how strongly the Q-function discriminates between candidates.

4 Relation to Prior Work

4.1 Curriculum Learning

Graves et al. (2017) use multi-armed bandits to select tasks; Jiang et al. (2018) train a separate "mentor" network to weight examples. Our approach differs by using the language model's own representations as features for the Q-function and applying soft Q-learning with temporal credit assignment, rather than treating selection as a stateless bandit problem.

4.2 Meta-Learning

MAML (Finn et al., 2017) learns initializations for fast adaptation. We share the computational structure—reasoning about the effect of gradient updates—but learn *what* to train on rather than *where* to start.

4.3 RL as Inference

The control-as-inference framework (Levine, 2018; Rawlik et al., 2013) casts optimal control as probabilistic inference, with the Q-function serving as an energy function and the optimal policy as the corresponding Boltzmann distribution. Our algorithm instantiates this framework in the curriculum selection setting: the Q-network provides unnormalized log-probabilities over actions, and the Boltzmann policy is derived analytically without requiring a separate policy network. This is the soft Q-learning approach of Haarnoja et al. (2017), applied to curriculum selection with a standard discount factor for Bellman contraction.

4.4 Soft Q-Learning

Soft Q-learning (Haarnoja et al., 2017) augments the standard Bellman equation with an entropy bonus, yielding a Boltzmann policy without requiring a separate policy network. SAC (Haarnoja et al., 2018) extends this to continuous actions with a learned temperature. Our setting is simpler: the action space is a finite set of N candidates, so the Boltzmann policy and soft value can be computed exactly via softmax and logsumexp. Following standard practice in deep Q-learning (Mnih et al., 2015; Lillicrap et al., 2016), we use a Polyak-averaged target network for the bootstrap value and normalise the reward by $(1 - \gamma)$ to keep Q-values on the same scale as the reward.

4.5 Intrinsic Motivation

Our objective relates to compression progress (Schmidhuber) and active inference, but avoids the memory requirements of the former and the dark room problem of the latter by using a fixed held-out set as a proxy for predictive success. The absolute log-probability reward (1) provides a direct measure of predictive fitness, connecting to the epistemic homeostasis motivation: the agent is penalised for *being* in a state of poor prediction, not merely for failing to improve.

4.6 Connection to Prior Work

This proposal builds directly on Markovian Transformers for Informative Language Modeling (`https://arxiv.org/abs/2404.18988`), which introduces a framework for training language models with RL to produce causally load-bearing Chain-of-Thought reasoning. Both projects share a common structure: use RL to learn intermediate representations that improve prediction on held-out data. In the Markovian Transformers work, the learned object is a CoT:

$$\text{Question} \rightarrow \text{CoT} \rightarrow \text{Answer}$$

In the current proposal, the learned object is a curriculum:

$$\text{Model State} \rightarrow \text{Selected Data} \rightarrow \text{Improved Predictions}$$

The Markovian Transformers work demonstrates that this general approach is tractable and yields large gains on reasoning benchmarks (e.g., GSM8K: 19.6% $\rightarrow$ 57.1%). The current proposal extends this framework from learning *what to say* to learning *what to study*.

5 Broader Motivation

Language models trained to predict text develop remarkable capabilities from a simple objective. Yet they require extensive post-training to behave agentically and arguably lack a kind of global coherence. One hypothesis: the training process is purely observational—the model never takes actions that affect what it observes.

This project is a stepping stone toward studying whether learned curriculum selection produces qualitatively different agents. The immediate goal is demonstrating sample efficiency gains. The longer-term question is whether controlling one's own learning process contributes to the coherence and agency that current models seem to lack.

5.1 Long-Term Direction: Formalizing Homeostasis

Biological agents are shaped by survival pressures. Hunger, pain, and fatigue are not arbitrary reward signals—they are tied to the organism's continued existence. Current approaches to intrinsic motivation (curiosity, empowerment, compression progress) capture aspects of adaptive behavior but lack this grounding in self-preservation.

A long-term goal of this research program is to formalize homeostasis and survival into a simple, biologically plausible metric that could serve as a foundation for intrinsic motivation in artificial systems. The current project—learning to select data that improves future prediction—is a minimal step in this direction: the agent takes actions that maintain its predictive capacity, a kind of epistemic homeostasis. The absolute reward formulation makes this connection explicit: the agent is directly penalised for poor predictive fitness at every moment, not merely for failing to improve.

Algorithm 1 Curriculum Selection via Normalised Discounted Soft Q-Learning

Initialise: LM parameters θ; Q-network parameters ϕ; target parameters $\bar{\phi} \leftarrow \phi$

// Bootstrap step (no Q update)
$C_0 \leftarrow \textsc{NextBatch}(\text{stream}, N)$
$H_0 \leftarrow \textsc{DetachedHidden}(\theta, C_0)$ ▷ $[N, d]$ — no grad into θ
$q_0 \leftarrow Q_\phi(H_0)$ ▷ $[N, 1]$
$a_0 \leftarrow \textsc{Sample}\big(\textsc{Softmax}(q_0/\beta)\big)$
$h_{\text{prev}} \leftarrow H_0[a_0]$ ▷ Store hidden state of selected action
$\theta \leftarrow \text{SGDStep}(\theta, C_0[a_0], \alpha, G)$ ▷ LM update, clipped
$r_{\text{prev}} \leftarrow \frac{1}{M} \log p_\theta(\hat{D}_0)$ ▷ Reward after update

for $k = 1, 2, \ldots$ **do** *// Main loop*
  $C_k \leftarrow \textsc{NextBatch}(\text{stream}, N)$
  $H_k \leftarrow \textsc{DetachedHidden}(\theta, C_k)$ ▷ Serves dual purpose below
  $q_k \leftarrow Q_\phi(H_k)$

  // Q update for previous transition (target network for bootstrap)
  $\bar{q}_k \leftarrow Q_{\bar{\phi}}(H_k)$ ▷ Target network Q-values, no gradient
  $V_k \leftarrow \beta \log \sum_{a'} \exp\big(\bar{q}_k[a']/\beta\big)$ ▷ Soft value from target network
  $y \leftarrow (1-\gamma)\, r_{\text{prev}} + \gamma\, V_k$ ▷ Normalised TD target, stop-gradient
  $\hat{q} \leftarrow Q_\phi(h_{\text{prev}})$ ▷ Re-evaluate previous action with current ϕ
  Update ϕ to minimise $\frac{1}{2}(\hat{q} - y)^2$, clipped to $\|\nabla\| \leq G$
  $\bar{\phi} \leftarrow \tau\, \phi + (1-\tau)\, \bar{\phi}$ ▷ Polyak update

  // Action selection for current step
  $a_k \leftarrow \textsc{Sample}\big(\textsc{Softmax}(q_k/\beta)\big)$
  $h_{\text{prev}} \leftarrow H_k[a_k]$

  // LM update and reward
  $\theta \leftarrow \text{SGDStep}(\theta, C_k[a_k], \alpha, G)$
  $r_k \leftarrow \frac{1}{M} \log p_\theta(\hat{D}_k)$ ▷ Fresh $\hat{D}_k \subset D$, reward after update
  $r_{\text{prev}} \leftarrow r_k$
end for