# 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 $\theta$, 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 $Q_\phi$ with parameters $\phi$, implemented as a 2-layer MLP ($d \to 32 \to 1$) reading the base model's residual stream (see Section ??)
- Boltzmann temperature $\beta > 0$ controlling exploration
- Learning rate $\alpha$ for the LM gradient step; learning rate $\eta$ for the Q-network update
- Discount factor $\gamma \in (0, 1)$ controlling the effective planning horizon
- 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 $\theta_k$, 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 ??).

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 $\theta_{k+1}$ 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 cumulative objective $\sum_k r_k$ is the integral of the learning curve: it penalises every step spent with poor predictions, rewarding curricula that reach high performance quickly. Equivalently, maximising $\sum_k r_k$ minimises cumulative regret $\sum_k (L^* - r_k)$ where $L^*$ is the best achievable log-probability.

## 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 $\theta_k$, 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 $\theta$).
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 $\sigma$ is ReLU.

The hidden state $h_x$ implicitly encodes both the model's current state (through $\theta_k$, which determines the representations) and the identity of the candidate (through the input $x$). By detaching $h_x$, we ensure that $\theta$ is updated only by the language modeling objective, while $\phi$ 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 Discounted Soft Bellman Equation

The Q-function satisfies the discounted soft Bellman equation:

$$Q(s_k, a_k) = r_k + \gamma\,\beta \log \sum_{a'} \exp\big(Q(s_{k+1}, a')/\beta\big),\tag{3}$$

where $\gamma \in (0, 1)$ is the discount factor. The discount serves two purposes: it ensures contraction of the Bellman operator (guaranteeing convergence of Q-learning with function approximation), and it defines an effective planning horizon of $\sim 1/(1 - \gamma)$ steps over which the Q-function evaluates the consequences of a curriculum choice.

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 $Q(s_k, a_k) = r_k + \gamma\, V(s_{k+1})$. At convergence, $Q(s, a)$ represents the discounted sum of future rewards from choosing $a$ in state $s$ and following the Boltzmann policy thereafter. With $\gamma = 0.99$, the effective horizon is $\sim 100$ steps: the Q-function learns which training examples will lead to the best held-out performance over the next $\sim 100$ gradient updates.

The discount factor bounds the Q-values: since $|r_k| \le R_{\max}$, we have $|Q| \le R_{\max}/(1 - \gamma)$. For per-token log-probabilities ($R_{\max} \approx 4$) and $\gamma = 0.99$, this gives $|Q| \le 400$, which is well within the representable range of the MLP.

## 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.

---

**Algorithm 1** Curriculum Selection via Discounted Soft Q-Learning

---

1: **Initialise:** LM parameters $\theta$; Q-network parameters $\phi$

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

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

14:      *// Q update for previous transition (uses current forward pass as bootstrap)*
15:      $V_k \leftarrow \beta \log \sum_{a'} \exp\big(q_k[a']/\beta\big)$        $\triangleright$ Soft value of current state
16:      $y \leftarrow r_{\text{prev}} + \gamma V_k$        $\triangleright$ TD target, stop-gradient
17:      $\hat{q} \leftarrow Q_\phi(h_{\text{prev}})$        $\triangleright$ Re-evaluate previous action with current $\phi$
18:      Update $\phi$ to minimise $\frac{1}{2}(\hat{q} - y)^2$, clipped to $\|\nabla\| \leq G$

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

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

---

**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. The candidate forward pass dominates; the MLP and reward computations are comparatively cheap.

**Staleness.** The Q update at step $k$ uses $q_k$ (computed before the $\phi$ update) for both the bootstrap target and action selection. This one-step staleness is standard in online Q-learning and negligible in practice given the small MLP learning rate $\eta$.

## 2.6 Soft Mixture Training Variant

In the base algorithm (Section **??**), 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:

$$\pi_i = \frac{\exp(Q_\phi(s_k, x_i)/\beta)}{\sum_{j=1}^{N} \exp(Q_\phi(s_k, x_j)/\beta)}, \qquad x_i \in C_k. \tag{5}$$

The LM loss becomes:

$$\mathcal{L}_{\mathrm{mix}}(\theta; C_k) = \sum_{i=1}^{N} \pi_i \, \ell(\theta, x_i), \tag{6}$$

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

$$\theta_{k+1} = \theta_k - \alpha \, \nabla_\theta \mathcal{L}_{\mathrm{mix}}(\theta_k; C_k). \tag{7}$$

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

$$\nabla_\theta \mathcal{L}_{\mathrm{mix}} = \mathbb{E}_{x \sim \pi(\cdot|s_k)}\big[\nabla_\theta \ell(\theta_k, x)\big]. \tag{8}$$

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 **??**: a discrete action $a_k$ is still sampled from the Boltzmann policy, its hidden state $h_{\mathrm{prev}}$ is stored, and the standard TD update is applied at the next step. The Q-function continues to satisfy the discounted soft Bellman equation (**??**). 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 $Q(s, a) = r + \gamma V(s')$ 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× 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:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \left[ -\mathbb{E}_{x \sim \pi}\big[Q_\phi(s, x)\big] + \beta \,\mathrm{KL}\big(\pi \,\|\, p_0\big) \right], \tag{9}$$

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:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \left[ -\mathbb{E}_{x \sim \pi}\big[Q_\phi(s, x)\big] - \beta \, H(\pi) \right]. \tag{10}$$

The first term encourages selecting high-value candidates; the second encourages exploration. The temperature $\beta$ 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 $\pi$ acts as a variational posterior over actions, and the free energy is the negative ELBO:

$$\log p(\text{optimal} \mid s) \;\geq\; \mathbb{E}_{x \sim \pi}\left[\frac{Q(s, x)}{\beta}\right] - \mathrm{KL}(\pi \,\|\, p_0). \tag{11}$$

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 $\beta$ 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. The discount factor $\gamma$ provides the standard Bellman contraction guarantee, ensuring Q-values remain bounded under function approximation.

## 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 (**??**) 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.