Topics in Reinforcement Learning: AlphaZero, ChatGPT, Neuro-Dynamic Programming, Model Predictive Control, Discrete Optimization

Arizona State University
Course CSE 691, Spring 2024
Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

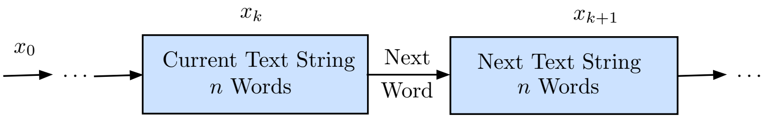
Dimitri P. Bertsekas dbertsek@asu.edu, Yuchao Li yuchaoli@asu.edu
March 20, 2024

Lecture 10
Rollout Algorithms for Most Likely Sequence Generation in *n*-Grams, Transformers, HMMs, and Markov Chains

Based on the paper
"Most Likely Sequence Generation for *n*-Grams, Transformers, HMMs, and Markov Chains by Using Rollout Algorithms", by Y. Li and D. Bertsekas, ArXiv, Mar. 2024

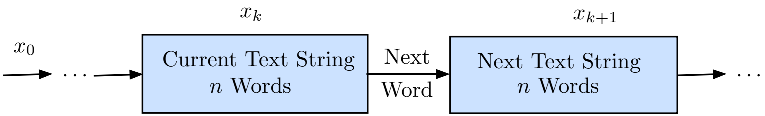
Outline

Recall the *n*-Gram Model of Next Word Generation



- One word added to the front and one word deleted from the back
- The *n*-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according to a policy of our choice
- Think of texting/next word suggestions; we can follow the suggested words or choose our own
- We focus on policies that produce highly likely sequences $\{x_1, x_2, \ldots, x_N\}$ starting from a given initial state/prompt x_0; a global measure of desirability

An Optimization Problem: Most Likely Sequence Selection Policy



- The most likely selection policy: Starting at x_0, select the most likely sequence $\{x_1, x_2, \ldots, x_N\}$, according to the n-gram's suggestions.
- This the one that maximizes

$$\text{Prob}(x_1, x_2, \ldots, x_N \mid x_0)$$

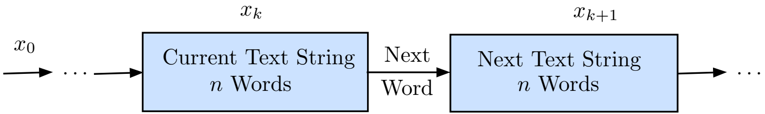
or equivalently maximizes

$$p(x_1 \mid x_0) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_2) \cdots p(X_N \mid x_{N-1})$$

[using the Markov property, i.e., $P(x_{k+1} \mid x_0, x_1, \ldots, x_k) = P(x_{k+1} \mid x_k)$ and the multiplication rule of conditional probability].

- We will view this policy as optimal/most desirable.
- Its advantage is that it plans into the future.
- We will use DP: (max product of rewards $\equiv$ max of sum of the reward logarithms)
- But DP requires intractable computation

We Will Look at Suboptimal Policies

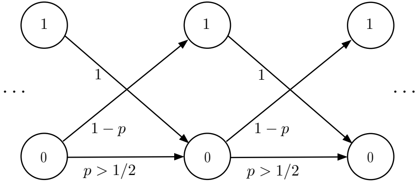


- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects x_{k+1} that maximizes the greedy Q-factor $Q(x_k, x_{k+1})$; i.e. the probability of the sequence

$$\text{Prob}(x_{k+1}, \text{Greedy sequence starting from } x_{k+1} \mid x_k)$$

- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.
- Double rollout: Rollout using the rollout-based-on-greedy (and its variants) as base policy.
- Under any one of these policies, the n-gram system is deterministic.
- As a result, we can contemplate powerful/sophisticated variants of rollout involving multistep lookahead (see Lecture 6, and Section 2.4 of the textbook).

Example: A 1-Gram with Vocabulary = $\{0, 1\}$



- Starting at $x_0 = 0$
- The greedy selection policy is: $\{x_0, x_1, x_2, \ldots\} = \{0, 0, 0, 0, \ldots\}$
- The most likely selection policy is: If $p^2 < 1 - p$ it selects $\{0, 1, 0, 1, \ldots\}$ [because $p^N < (1-p)^{N/2}$]; otherwise it selects $\{0, 0, 0, \ldots\}$
- The rollout selection policy with one-step lookahead, starting from $x_0 = 0$, compares two Q-factors corresponding to the two next states $x_1 = 0$ and $x_1 = 1$.
- If $p^2 < 1 - p$ it selects $x_1 = 1$; otherwise it selects $x_1 = 0$. Thus it generates the same sequence as the most likely selection policy.

An n-gram with its probabilities $p(x_{k+1} \mid x_k)$ defines a Markov chain (prob. of next state depends on the past-states history only through the current state):

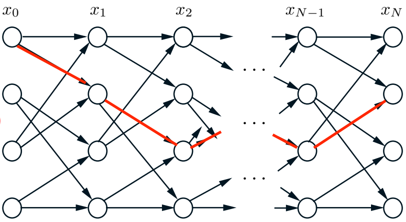
$$p(x_{k+1} \mid x_0, x_1, \ldots, x_k) = p(x_{k+1} \mid x_k)$$

An n-gram with vocabulary consisting of q different words involves q^n states. The state space can be enormous!

Inference in Hidden Markov Models: A Huge Class of Mathematically Equivalent Problems

Transition probabilities are "modified" by data

Replaced by $p(x_{k+1}, \text{data} \mid x_k)$

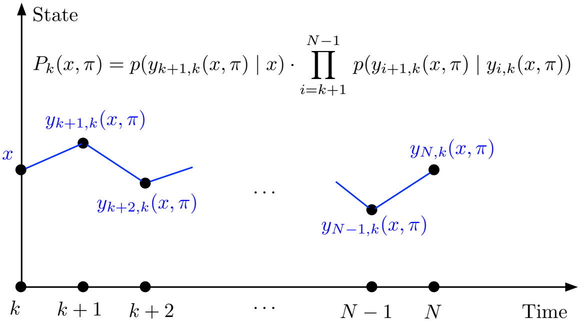


Find the most likely path (conditioned on the data)

Many applications: Speech recognition, language translation, computational linguistics, coding and error correction, bioinformatics, etc

Example: Given sentence (data), e.g., "He saw a beautiful fish in the water." Label each word as noun, pronoun, verb, adjective, adverb, determiner, etc.

- Often solved by a specialized form of DP, the Viterbi algorithm (1960s).
- For large state spaces exact solution is intractable and suboptimal shortest path-type algorithms have been used.
- Our DP-based rollout algorithms fully apply. The transition probabilities are replaced by data-dependent/time-dependent "weights"



Given a Markov chain with transition probabilities $p(x_{k+1} \mid x_k)$

- A selection policy π is a sequence of functions $\{\mu_0, \ldots, \mu_{N-1}\}$, which given the current state x_k, determines the next state x_{k+1} as $x_{k+1} = \mu_k(x_k)$.
- Given π and a starting state x at time k, the future states are denoted $y_{m,k}(x,\pi) =$ state at time $m > k$ starting at state x and using π
- The probability of its occurence (the reward-to-go function) is

$$P_k(x,\pi) = p\big(y_{k+1,k}(x,\pi) \mid x\big) \cdot \prod_{i=k+1}^{N-1} p\big(y_{i+1,k}(x,\pi) \mid y_{i,k}\big)$$

Most Likely and Greedy Selection Policies

- The most likely selection policy, denoted by $\pi^* = \{\mu_0^*, \dots, \mu_{N-1}^*\}$, maximizes over all policies π the probabilities $P_k(x, \pi)$ for every initial state x and time k:

$$P_k^*(x) = P_k(x, \pi^*) = \max_\pi P_k(x, \pi).$$

- DP-like algorithm to obtain π^* and its probabilities $P_k^*(x)$:
  - First compute the probabilities $P_k^*(x)$ backwards, for all x, according to

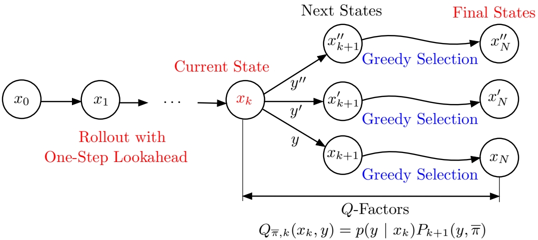
$$P_k^*(x) = \max_y p(y \mid x) P_{k+1}^*(y), \qquad k = N-1, \dots, 0,$$

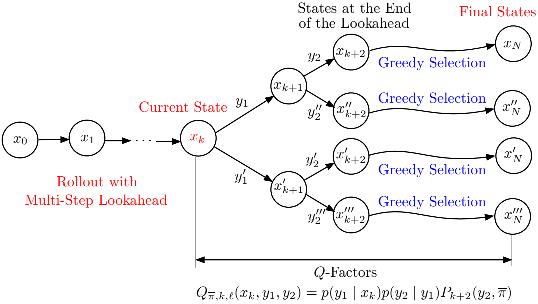
    starting with $P_N^*(x) \equiv 1$.
  - Then generate sequentially the selections $x_1^*, \dots, x_N^*$ of π^* forwards, according to

$$x_{k+1}^* = \mu_k^*(x_k^*) \in \arg\max_y p(y \mid x_k^*) P_{k+1}^*(y),$$

    starting with $x_0^* = x_0$.
- It is equivalent to the usual DP for multistage additive costs, after we take logarithms of the multiplicative expressions defining the probabilities $P_k(x, \pi)$.
- The greedy policy $\overline{\pi} = \{\overline{\mu}, \overline{\mu}, \dots, \overline{\mu}\}$ produces the next state by maximization of the corresponding transition probability over all y: $\overline{\mu}(x) = \arg\max_y p(y \mid x_k)$ (ties are broken according to a fixed rule for sequential consistency).

One-Step and Multistep Rollout Selection Policies





There are also truncated and simplified variants, etc

Induction proof of performance improvement for rollout with one-step lookahead. (Sequential consistency holds here; the base heuristic is a policy)

- Want to show that for the greedy policy $\overline{\pi}$ and the rollout policy $\tilde{\pi}$, we have

$$P_k(x, \overline{\pi}) \le P_k(x, \tilde{\pi}), \qquad \text{for all } x \text{ and } k$$

- For $k = N$ this holds, since we have $P_N(x, \overline{\pi}) = P_N(x, \tilde{\pi}) \equiv 1$.
- Assuming that

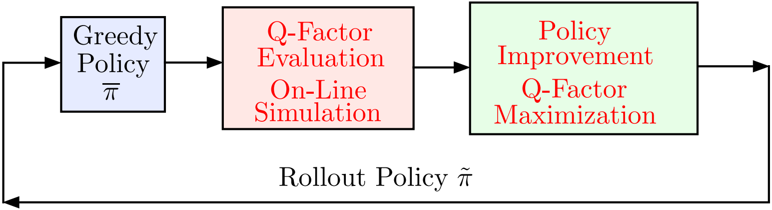
$$P_{k+1}(x, \overline{\pi}) \le P_{k+1}(x, \tilde{\pi}), \qquad \text{for all } x,$$

we will show that

$$P_k(x, \overline{\pi}) \le P_k(x, \tilde{\pi}), \qquad \text{for all } x.$$

- We have

$$\begin{aligned}
P_k(x, \tilde{\pi}) &= p\big(\tilde{\mu}_k(x) \mid x\big) P_{k+1}\big(\tilde{\mu}_k(x), \tilde{\pi}\big) \ \ \text{(by definition)} && (1)\\
&\ge p\big(\tilde{\mu}_k(x) \mid x\big) P_{k+1}\big(\tilde{\mu}_k(x), \overline{\pi}\big) \ \ \text{(by the induction hypothesis)} && (2)\\
&\ge p\big(\overline{\mu}_k(x) \mid x\big) P_{k+1}\big(\overline{\mu}_k(x), \overline{\pi}\big) \ \ \text{(rollout maximizes greedy Q-factor)} && (3)\\
&= P_k(x, \overline{\pi}) \ \ \text{(by definition)} && (4)
\end{aligned}$$



- Single rollout is rollout with greedy as base policy
- Double rollout is rollout with single rollout as base policy
- Triple rollout is rollout with double rollout as base policy
- k-order rollout is rollout with $(k-1)$-order rollout as base policy
- These rollout algorithms can be multistep lookahead, truncated, simplified, etc
- For double rollout, at any encountered state x_k, we run the single rollout from every next state y and select x_{k+1} to maximize the Q-factor $Q_{\tilde{\pi},k}(x_k, y)$ over y
- Double rollout requires $O(q \cdot N)$ applications of single rollout (q is the number of possible next states after simplification)

With a large enough number of policy iterations, the most likely sequence is obtained

Computational Experiments with Markov Chains

States	1	2	$\cdots$	100
1	$p(1 \mid 1)$	$p(2 \mid 1)$	$\cdots$	$p(100 \mid 1)$
2	$p(1 \mid 2)$	$p(2 \mid 2)$	$\cdots$	$p(100 \mid 2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
100	$p(1 \mid 100)$	$p(2 \mid 100)$	$\cdots$	$p(100 \mid 100)$

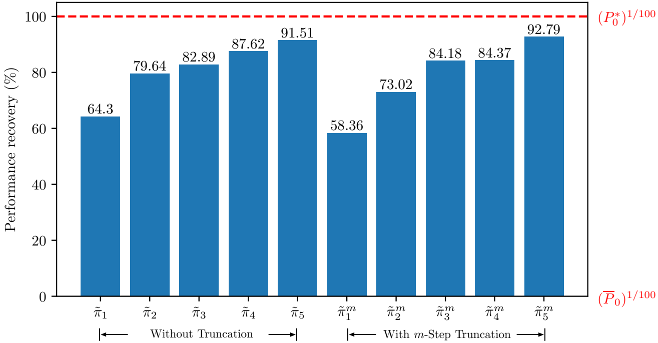
States $1, 2, \ldots, 100$
xth row: transition probabilities from x to all other states

- We consider $N = 100$ most likely sequence for a set C of Markov chains involving 100 states. For each state x, there are 5 states y such that $p(y \mid x) > 0$
- The transition probabilities are given in a lookup table
- The most likely sequence can be computed via DP
- We measure the performance of rollout by the percentage recovery of optimality loss of the greedy policy, given by

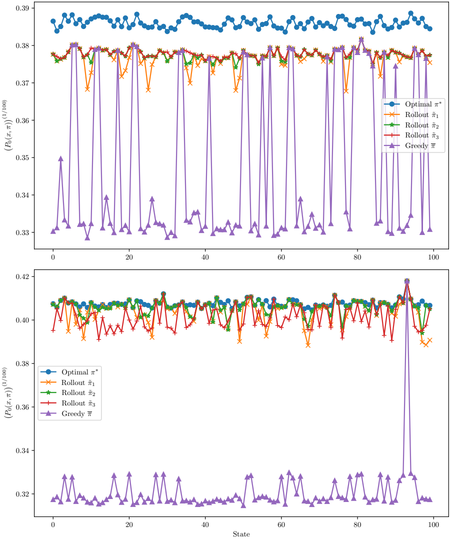
$$\frac{(\tilde{P}_0)^{1/N} - (\overline{P}_0)^{1/N}}{(P_0^*)^{1/N} - (\overline{P}_0)^{1/N}} \times 100\,(\%)$$

where $(\overline{P}_0)^{1/N}$, $(\tilde{P}_0)^{1/N}$, and $(P_0^*)^{1/N}$ are the average transition probabilities under $\overline{\pi}$, $\tilde{\pi}$, and π^*, respectively.

Percentage Recovery by Rollout

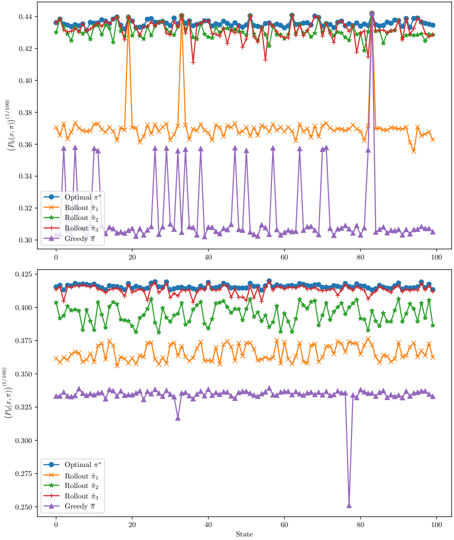


- We consider rollout with one-step and ℓ-step lookahead ($\ell = 2$ to 5), denoted by $\tilde{\pi}_\ell$, and their m-step truncated counterparts with $m = 10$, denoted by $\tilde{\pi}_\ell^m$
- General observations from the experiments:
  - Rollout leads to substantial improvements over the greedy policy in all test cases
  - Longer lookahead results in improvement on average
  - The performance seems unaffected by the 90% truncation of the rollout horizon



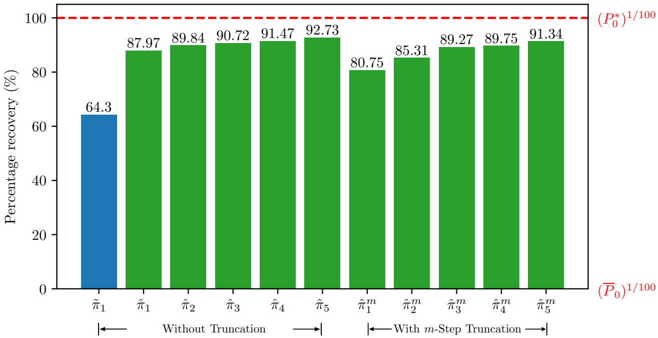
For a given Markov chain, $\big(P_0(x,\pi)\big)^{1/N}$ with π being π^*, $\tilde{\pi}_\ell$ with $\ell = 1, 2, 3$, or $\overline{\pi}$

Typical Patterns of Rollout (2)



For a given Markov chain, $\big(P_0(x,\pi)\big)^{1/N}$ with π being π^*, $\tilde{\pi}_\ell$ with $\ell = 1, 2, 3$, or $\overline{\pi}$

Percentage Recovery by Double Rollout

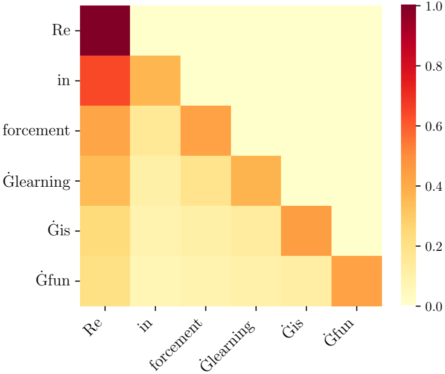


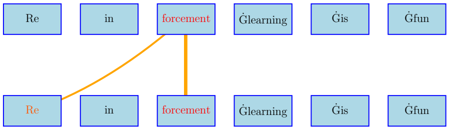
- We consider double rollout with one-step and ℓ-step lookahead ($\ell = 2$ to 5), denoted by $\hat{\pi}_\ell$, and their m-step truncated counterparts with $m = 10$, denoted by $\hat{\pi}_\ell^m$, and their results are shown in green bars
- General observations from the experiments:
  - Double rollout algorithm and its variants lead to significant performance improvement over both the greedy policy and (single) rollout with one-step lookahead
  - The truncated versions of double rollout remain effective, despite large computational savings

Generative Pre-Trained Transformer - GPT

- Transformer architectures improve in important ways on the earlier neural networks by using the attention mechanism
- Such neural networks are often Pre-Trained with a lot of general purpose data before they are further trained (fine-tuned) with special purpose data
- Owing to both the architecture and pre-training, the resulting neural networks are Generative: being able to create new content and to perform open-ended tasks
- Tremendous amount of applications: natural language processing, computer vision, image/music generation (see the figure above) ...
- We focus on text generation and view the GPT as an n-gram.

Attention Mechanism in a GPT

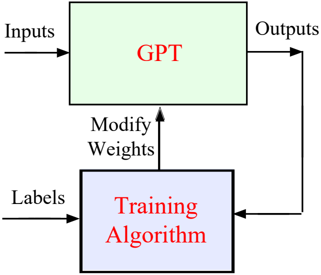




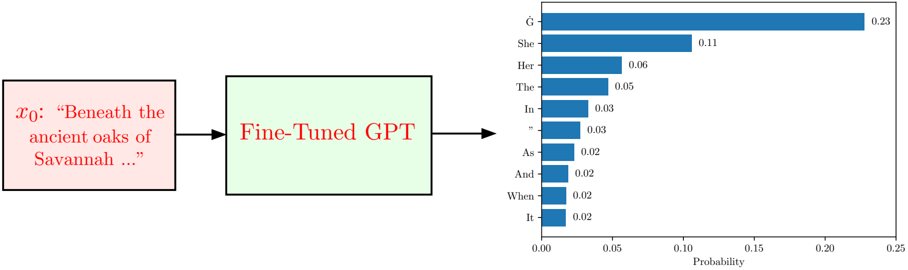
- Consider the text string ‘Reinforcement learning is fun’ as an input
- It is viewed as 6 ‘words’ (called tokens) by the GPT
- For each feasible ordered ‘word’ pair, an attention score is generated, measuring the affinity between them: how strongly they are related
- The ‘attention’ of the neural network is given to relations with high scores

Training Data and Algorithms for GPT

Inputs	Labels
Re	in
Re in	forcement
Re in forcement	Ġlearning
Re in forcement Ġlearning	Ġis
Re in forcement Ġlearning Ġis	Ġfun

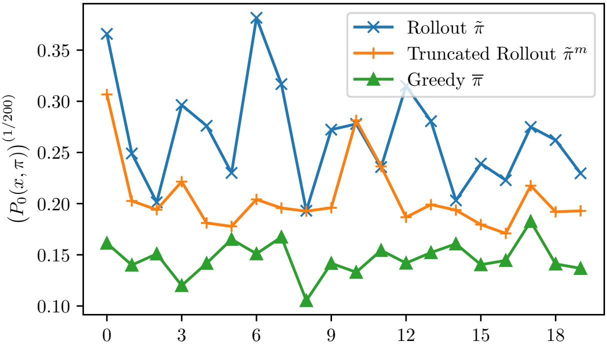


- The text string ‘Reinforcement learning is fun’ corresponds to 5 input-label pairs used for training
- Using text strings for training does not require explicit data-labeling.
- The automatic generation of labeled data from unlabeled data for training enables the pre-training with large amount of general purpose data
- The labeled data generation and training process are collectively known as self-supervised learning
- The training algorithms used for GPT are scaled versions of stochastic gradient descent, more on this in the next lecture



- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated Markov chain. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the transformer
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- The large vocabulary size leads to excessive Q-factor computations
- We applied simplified rollout and its truncated counterpart
- Rollout can take advantage of the parallel processing power of graphical processing units (GPU)

Performance of Simplified Rollout



- We applied two simplification techniques:
  - Computing only 10 Q-factors corresponding to top ten most likely next words: simplified rollout with one-step lookahead
  - In addition, truncating the simulation after 10 steps: *m*-step truncated rollout
- General observations from the experiments:
  - Simplified rollout has substantial improvement over the greedy policy, with modest computation increase
  - The truncated counterpart still improves upon the greedy policy in all our test cases

Neural network and other approximation architectures. Off-line training and uses in RL contexts. See Chapter 3 of the textbook.