Applications of ‘Model Predictive Control’ in Artificial Intelligence

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

Based on the papers
“Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming", by D.P. Bertsekas, arXiv:2406.00592, Jun. 2024
“Most Likely Sequence Generation for *n*-Grams, Transformers, HMMs, and Markov Chains by Using Rollout Algorithms", by Y. Li and D.P. Bertsekas, arXiv:2403.15465, Mar. 2024
“An Approximate Dynamic Programming Framework for Occlusion-Robust Multi-Object Tracking", by P. Musunuru, Y. Li, J. Weber, and D.P. Bertsekas, arXiv:2405.15137, May 2024

Outline

Outline

1. Model Predictive Control as Approximation in Value Space
2. Computing Most Likely Sequence of a Language Model
3. Addressing Multiple Object Tracking/Data Association Problem
4. Approximation in Value Space with Fine-Tuned Language Model (if time permits)

Outline

1. Model Predictive Control as Approximation in Value Space
2. Computing Most Likely Sequence of a Language Model
3. Addressing Multiple Object Tracking/Data Association Problem
4. Approximation in Value Space with Fine-Tuned Language Model (if time permits)

Outline

1. Model Predictive Control as Approximation in Value Space
2. Computing Most Likely Sequence of a Language Model
3. Addressing Multiple Object Tracking/Data Association Problem
4. Approximation in Value Space with Fine-Tuned Language Model (if time permits)

Outline

Model Predictive Control and AlphaGo/AlphaZero

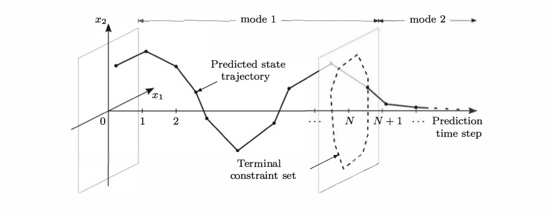


Figure: Modified from [Kouvaritakis & Cannon, Fig. 2.1]

- Model predictive control (MPC): Select control at the present time based on the prediction and evaluation of state trajectories into the future
- The predicted trajectories are truncated after finite stages, and an offline computed function and/or constraint are used at the end of the prediction for evaluation.
- The prediction and evaluation is carried out online: repeated at each stage
- AlphaGo/AlphaZero: Highly similar structures involving prediction and evaluation of future board configuration, using trained neural networks
- Can we connect MPC and AlphaGo/AlphaZero via a unifying framework?

Model Predictive Control and AlphaGo/AlphaZero

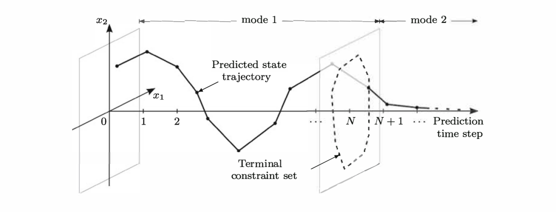


Figure: Modified from [Kouvaritakis & Cannon, Fig. 2.1]

- Model predictive control (MPC): Select control at the present time based on the prediction and evaluation of state trajectories into the future
- The predicted trajectories are truncated after finite stages, and an offline computed function and/or constraint are used at the end of the prediction for evaluation.
- The prediction and evaluation is carried out online: repeated at each stage
- AlphaGo/AlphaZero: Highly similar structures involving prediction and evaluation of future board configuration, using trained neural networks
- Can we connect MPC and AlphaGo/AlphaZero via a unifying framework?

Model Predictive Control and AlphaGo/AlphaZero

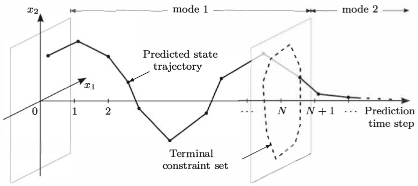
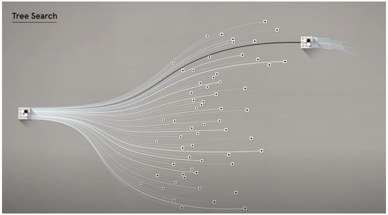


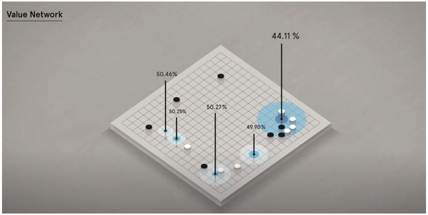
Figure: Modified from [Kouvaritakis & Cannon, Fig. 2.1]

- Model predictive control (MPC): Select control at the present time based on the prediction and evaluation of state trajectories into the future
- The predicted trajectories are truncated after finite stages, and an offline computed function and/or constraint are used at the end of the prediction for evaluation.
- The prediction and evaluation is carried out online: repeated at each stage
- AlphaGo/AlphaZero: Highly similar structures involving prediction and evaluation of future board configuration, using trained neural networks
- Can we connect MPC and AlphaGo/AlphaZero via a unifying framework?

Model Predictive Control and AlphaGo/AlphaZero



(a) Tree search

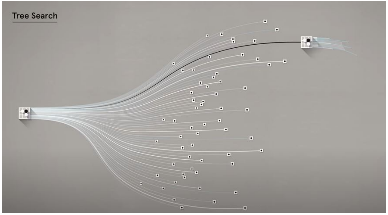


(b) Value network

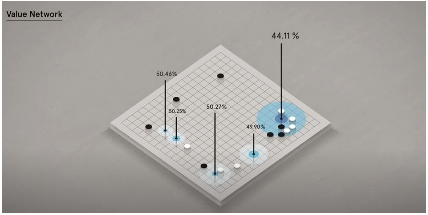
Figure: From AlphaGo movie

- Model predictive control (MPC): Select control at the present time based on the prediction and evaluation of state trajectories into the future
- The predicted trajectories are truncated after finite stages, and an offline computed function and/or constraint are used at the end of the prediction for evaluation.
- The prediction and evaluation is carried out online: repeated at each stage
- AlphaGo/AlphaZero: Highly similar structures involving prediction and evaluation of future board configuration, using trained neural networks
- Can we connect MPC and AlphaGo/AlphaZero via a unifying framework?

Model Predictive Control and AlphaGo/AlphaZero



(a) Tree search

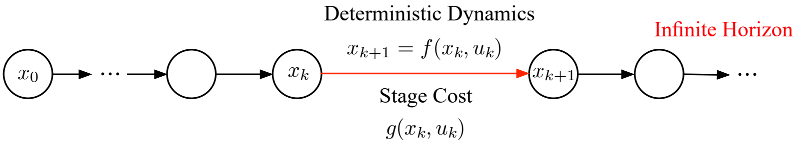


(b) Value network

Figure: From AlphaGo movie

- Model predictive control (MPC): Select control at the present time based on the prediction and evaluation of state trajectories into the future
- The predicted trajectories are truncated after finite stages, and an offline computed function and/or constraint are used at the end of the prediction for evaluation.
- The prediction and evaluation is carried out online: repeated at each stage
- AlphaGo/AlphaZero: Highly similar structures involving prediction and evaluation of future board configuration, using trained neural networks
- Can we connect MPC and AlphaGo/AlphaZero via a unifying framework?

Dynamic Programming Model



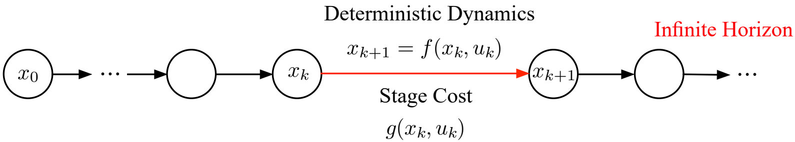
- The state space X and the control constraint set $U(x) \subset U$.
- The system dynamics $f : X \times U \to X$ and the stage cost $g : X \times U \to \Re^*$.
- A policy $\mu : X \to U$ with $\mu(x) \in U(x)$ for all x and its cost function

$$J_\mu(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, \mu(x_k)).$$

- The optimal cost function $J^* : X \to \Re$ and optimal policy $\mu^* : X \to U$:

$$J^*(x_0) = \min_{u_k, k=0,1,\ldots} \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, u_k), \quad J_{\mu^*}(x) = J^*(x).$$

Dynamic Programming Model



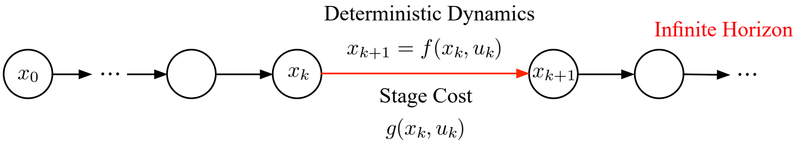
- The state space X and the control constraint set $U(x) \subset U$.
- The system dynamics $f : X \times U \to X$ and the stage cost $g : X \times U \to \Re^*$.
- A policy $\mu : X \to U$ with $\mu(x) \in U(x)$ for all x and its cost function

$$J_\mu(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} g\big(x_k, \mu(x_k)\big).$$

- The optimal cost function $J^* : X \to \Re$ and optimal policy $\mu^* : X \to U$:

$$J^*(x_0) = \min_{u_k, k=0,1,\ldots} \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, u_k), \quad J_{\mu^*}(x) = J^*(x).$$

Dynamic Programming Model



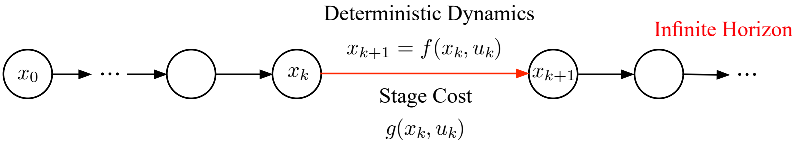
- The state space X and the control constraint set $U(x) \subset U$.
- The system dynamics $f : X \times U \to X$ and the stage cost $g : X \times U \to \Re^*$.
- A policy $\mu : X \to U$ with $\mu(x) \in U(x)$ for all x and its cost function

$$J_\mu(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} g\big(x_k, \mu(x_k)\big).$$

- The optimal cost function $J^* : X \to \Re$ and optimal policy $\mu^* : X \to U$:

$$J^*(x_0) = \min_{u_k, k=0,1,\ldots} \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, u_k), \quad J_{\mu^*}(x) = J^*(x).$$

Dynamic Programming Model

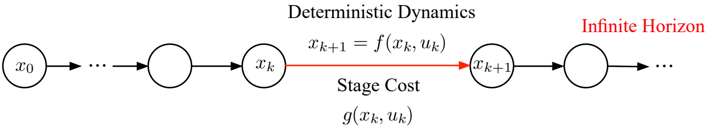


- The state space X and the control constraint set $U(x) \subset U$.
- The system dynamics $f : X \times U \to X$ and the stage cost $g : X \times U \to \Re^*$.
- A policy $\mu : X \to U$ with $\mu(x) \in U(x)$ for all x and its cost function

$$J_\mu(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} g\big(x_k, \mu(x_k)\big).$$

- The optimal cost function $J^* : X \to \Re$ and optimal policy $\mu^* : X \to U$:

$$J^*(x_0) = \min_{u_k, k=0,1,\dots} \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, u_k), \quad J_{\mu^*}(x) = J^*(x).$$



- The optimal cost function J^* fulfills Bellman’s equation:

$$J^*(x) = \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}, \quad \text{for all } x.$$

- The optimization problem is transformed to solving fixed point equation:

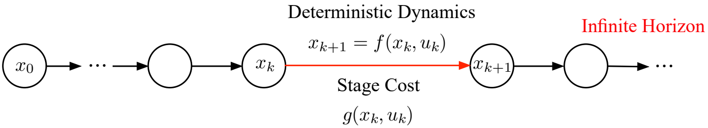
$$\min_{u_k, k=0,1,\ldots} \lim_{N \to \infty} \sum_{k=0}^{N-1} g(x_k, u_k) \implies J^*(x) = \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}.$$

- Upon obtaining J^*, the optimal policy μ^* can be computed via:

$$\mu^*(x) \in \arg \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}$$

- The sum $g(x,u) + J^*\big(f(x,u)\big)$ is known as the Q-factor, and denoted by $Q(x,u)$.

Bellman's Equation



- The optimal cost function J^* fulfills Bellman's equation:

$$J^*(x) = \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}, \quad \text{for all } x.$$

- The optimization problem is transformed to solving fixed point equation:

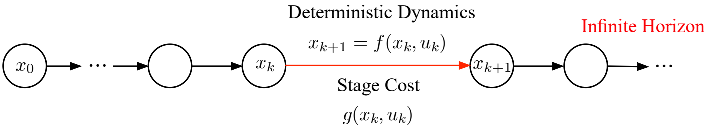
$$\min_{u_k, k=0,1,\ldots} \lim_{N \to \infty} \sum_{k=0}^{N-1} g(x_k, u_k) \implies J^*(x) = \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}.$$

- Upon obtaining J^*, the optimal policy μ^* can be computed via:

$$\mu^*(x) \in \arg \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}$$

- The sum $g(x,u) + J^*\big(f(x,u)\big)$ is known as the Q-factor, and denoted by $Q(x,u)$.

Bellman's Equation



- The optimal cost function J^* fulfills Bellman's equation:

$$J^*(x) = \min_{u \in U(x)} \big\{ g(x,u) + J^*\big(f(x,u)\big) \big\}, \quad \text{for all } x.$$

- The optimization problem is transformed to solving fixed point equation:

$$\min_{u_k, k=0,1,\ldots} \lim_{N\to\infty} \sum_{k=0}^{N-1} g(x_k, u_k) \implies J^*(x) = \min_{u \in U(x)} \big\{ g(x,u) + J^*\big(f(x,u)\big) \big\}.$$

- Upon obtaining J^*, the optimal policy μ^* can be computed via:

$$\mu^*(x) \in \arg\min_{u \in U(x)} \big\{ g(x,u) + J^*\big(f(x,u)\big) \big\}$$

- The sum $g(x,u) + J^*\big(f(x,u)\big)$ is known as the Q-factor, and denoted by $Q(x,u)$.

Approximation in Value Space: Basic Form

- Typically, it is intractable to compute the optimal cost function J^*. As a result, the optimal policy μ^* cannot be computed via

$$\mu^*(x) \in \arg\min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}.$$

- Approximation in value space: replacing J^* with some function $\tilde{J}$ that is obtained through offline training, and apply the policy $\tilde{\mu}$ obtained through online play:

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} \left\{ g(x,u) + \tilde{J}\big(f(x,u)\big) \right\}. \quad (1)$$

  The form is called one-step lookahead.

- The offline computation ensures that the values $\tilde{J}(x)$ are 'known' for all x.
- The online computation (1) for $\tilde{\mu}(x)$ is only for the state x that we encounter.
- Why effective: Computing J^* can be viewed as a root finding problem:

$$J^*(x) - \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\} = 0, \quad \text{for all } x. \quad (2)$$

- Approximation in value space is one step of Netwon's method for solving (2), with the offline computed $\tilde{J}$ as the initial guess of J^*.

Approximation in Value Space: Basic Form

- Typically, it is intractable to compute the optimal cost function J^*. As a result, the optimal policy μ^* cannot be computed via

$$\mu^*(x) \in \arg \min_{u \in U(x)} \{g(x,u) + J^*\big(f(x,u)\big)\}.$$

- Approximation in value space: replacing J^* with some function $\tilde{J}$ that is obtained through offline training, and apply the policy $\tilde{\mu}$ obtained through online play:

$$\tilde{\mu}(x) \in \arg \min_{u \in U(x)} \{g(x,u) + \tilde{J}\big(f(x,u)\big)\}. \tag{1}$$

  The form is called one-step lookahead.

- The offline computation ensures that the values $\tilde{J}(x)$ are 'known' for all x.
- The online computation (1) for $\tilde{\mu}(x)$ is only for the state x that we encounter.
- Why effective: Computing J^* can be viewed as a root finding problem:

$$J^*(x) - \min_{u \in U(x)} \{g(x,u) + J^*\big(f(x,u)\big)\} = 0, \quad \text{for all } x. \tag{2}$$

- Approximation in value space is one step of Netwon's method for solving (2), with the offline computed $\tilde{J}$ as the initial guess of J^*.

Approximation in Value Space: Basic Form

- Typically, it is intractable to compute the optimal cost function J^*. As a result, the optimal policy μ^* cannot be computed via

$$\mu^*(x) \in \arg\min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}.$$

- Approximation in value space: replacing J^* with some function $\tilde{J}$ that is obtained through offline training, and apply the policy $\tilde{\mu}$ obtained through online play:

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} \left\{ g(x,u) + \tilde{J}\big(f(x,u)\big) \right\}. \qquad (1)$$

  The form is called one-step lookahead.

- The offline computation ensures that the values $\tilde{J}(x)$ are 'known' for all x.
- The online computation (1) for $\tilde{\mu}(x)$ is only for the state x that we encounter.
- Why effective: Computing J^* can be viewed as a root finding problem:

$$J^*(x) - \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\} = 0, \quad \text{for all } x. \qquad (2)$$

- Approximation in value space is one step of Netwon's method for solving (2), with the offline computed $\tilde{J}$ as the initial guess of J^*.

Approximation in Value Space: Basic Form

- Typically, it is intractable to compute the optimal cost function J^*. As a result, the optimal policy μ^* cannot be computed via

$$\mu^*(x) \in \arg \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\}.$$

- Approximation in value space: replacing J^* with some function $\tilde{J}$ that is obtained through offline training, and apply the policy $\tilde{\mu}$ obtained through online play:

$$\tilde{\mu}(x) \in \arg \min_{u \in U(x)} \left\{ g(x,u) + \tilde{J}\big(f(x,u)\big) \right\}. \quad (1)$$

  The form is called one-step lookahead.
- The offline computation ensures that the values $\tilde{J}(x)$ are ‘known’ for all x.
- The online computation (1) for $\tilde{\mu}(x)$ is only for the state x that we encounter.
- Why effective: Computing J^* can be viewed as a root finding problem:

$$J^*(x) - \min_{u \in U(x)} \left\{ g(x,u) + J^*\big(f(x,u)\big) \right\} = 0, \quad \text{for all } x. \quad (2)$$

- Approximation in value space is one step of Netwon’s method for solving (2), with the offline computed $\tilde{J}$ as the initial guess of J^*.

Approximation in Value Space: Basic Form

- Typically, it is intractable to compute the optimal cost function J^*. As a result, the optimal policy μ^* cannot be computed via

$$\mu^*(x) \in \arg\min_{u \in U(x)} \big\{ g(x,u) + J^*\big(f(x,u)\big) \big\}.$$

- Approximation in value space: replacing J^* with some function $\tilde{J}$ that is obtained through offline training, and apply the policy $\tilde{\mu}$ obtained through online play:

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} \big\{ g(x,u) + \tilde{J}\big(f(x,u)\big) \big\}. \tag{1}$$

  The form is called one-step lookahead.

- The offline computation ensures that the values $\tilde{J}(x)$ are 'known' for all x.
- The online computation (1) for $\tilde{\mu}(x)$ is only for the state x that we encounter.
- Why effective: Computing J^* can be viewed as a root finding problem:

$$J^*(x) - \min_{u \in U(x)} \big\{ g(x,u) + J^*\big(f(x,u)\big) \big\} = 0, \quad \text{for all } x. \tag{2}$$

- Approximation in value space is one step of Netwon's method for solving (2), with the offline computed $\tilde{J}$ as the initial guess of J^*.

Approximation in Value Space: Variants

- Rollout: using cost function J_μ of a policy μ as $\tilde{J}$, where μ is called a base policy
- Truncated rollout: setting $\tilde{J} \approx J_\mu$, e.g., after computing some $\hat{J}$, define $\tilde{J}(x)$ as

$$\tilde{J}(x_\ell) = \sum_{k=\ell}^{\ell+m-1} g(x_k, \mu(x_k)) + \hat{J}(x_{\ell+m})$$

- ℓ-step lookahead: optimizing over ℓ controls $u_0, u_1, \dots, u_{\ell-1}$:

$$\tilde{\mu}(x_0) \in \arg \min_{u_0 \in U(x_0)} \Big(g(x_0, u_0) + \min_{u_k,\, k=1,\dots,\ell-1} \big(\sum_{k=1}^{\ell-1} g(x_k, u_k) + \tilde{J}(x_\ell) \big) \Big).$$

- Simplified minimization: construct a subset $\bar{U}(x) \subset U(x)$, and compute $\tilde{\mu}(x)$ via

$$\tilde{\mu}(x) \in \arg \min_{u \in \bar{U}(x)} \big\{ g(x, u) + \tilde{J}(f(x, u)) \big\}.$$

- All the ideas discussed thus far apply to finite horizon problems!

Approximation in Value Space: Variants

- Rollout: using cost function J_μ of a policy μ as $\tilde{J}$, where μ is called a base policy
- Truncated rollout: setting $\tilde{J} \approx J_\mu$, e.g., after computing some $\hat{J}$, define $\tilde{J}(x)$ as

$$\tilde{J}(x_\ell) = \sum_{k=\ell}^{\ell+m-1} g(x_k, \mu(x_k)) + \hat{J}(x_{\ell+m})$$

- ℓ-step lookahead: optimizing over ℓ controls $u_0, u_1, \ldots, u_{\ell-1}$:

$$\tilde{\mu}(x_0) \in \arg \min_{u_0 \in U(x_0)} \Big(g(x_0, u_0) + \min_{u_k,\, k=1,\ldots,\ell-1} \big(\sum_{k=1}^{\ell-1} g(x_k, u_k) + \tilde{J}(x_\ell) \big) \Big).$$

- Simplified minimization: construct a subset $\bar{U}(x) \subset U(x)$, and compute $\tilde{\mu}(x)$ via

$$\tilde{\mu}(x) \in \arg \min_{u \in \bar{U}(x)} \big\{ g(x,u) + \tilde{J}(f(x,u)) \big\}.$$

- All the ideas discussed thus far apply to finite horizon problems!

Approximation in Value Space: Variants

- Rollout: using cost function J_μ of a policy μ as $\tilde{J}$, where μ is called a base policy
- Truncated rollout: setting $\tilde{J} \approx J_\mu$, e.g., after computing some $\hat{J}$, define $\tilde{J}(x)$ as

$$\tilde{J}(x_\ell) = \sum_{k=\ell}^{\ell+m-1} g(x_k, \mu(x_k)) + \hat{J}(x_{\ell+m})$$

- ℓ-step lookahead: optimizing over ℓ controls $u_0, u_1, \ldots, u_{\ell-1}$:

$$\tilde{\mu}(x_0) \in \arg \min_{u_0 \in U(x_0)} \Big(g(x_0, u_0) + \min_{u_k,\, k=1,\ldots,\ell-1} \big(\sum_{k=1}^{\ell-1} g(x_k, u_k) + \tilde{J}(x_\ell) \big) \Big).$$

- Simplified minimization: construct a subset $\bar{U}(x) \subset U(x)$, and compute $\tilde{\mu}(x)$ via

$$\tilde{\mu}(x) \in \arg \min_{u \in \bar{U}(x)} \big\{ g(x, u) + \tilde{J}(f(x, u)) \big\}.$$

- All the ideas discussed thus far apply to finite horizon problems!

Approximation in Value Space: Variants

- Rollout: using cost function J_μ of a policy μ as $\tilde{J}$, where μ is called a base policy
- Truncated rollout: setting $\tilde{J} \approx J_\mu$, e.g., after computing some $\hat{J}$, define $\tilde{J}(x)$ as

$$\tilde{J}(x_\ell) = \sum_{k=\ell}^{\ell+m-1} g(x_k, \mu(x_k)) + \hat{J}(x_{\ell+m})$$

- ℓ-step lookahead: optimizing over ℓ controls $u_0, u_1, \ldots, u_{\ell-1}$:

$$\tilde{\mu}(x_0) \in \arg \min_{u_0 \in U(x_0)} \Big(g(x_0, u_0) + \min_{u_k,\, k=1,\ldots,\ell-1} \big(\sum_{k=1}^{\ell-1} g(x_k, u_k) + \tilde{J}(x_\ell) \big) \Big).$$

- Simplified minimization: construct a subset $\bar{U}(x) \subset U(x)$, and compute $\tilde{\mu}(x)$ via

$$\tilde{\mu}(x) \in \arg \min_{u \in \bar{U}(x)} \big\{ g(x, u) + \tilde{J}\big(f(x, u)\big) \big\}.$$

- All the ideas discussed thus far apply to finite horizon problems!

Approximation in Value Space: Variants

- Rollout: using cost function J_μ of a policy μ as $\tilde{J}$, where μ is called a base policy
- Truncated rollout: setting $\tilde{J} \approx J_\mu$, e.g., after computing some $\hat{J}$, define $\tilde{J}(x)$ as

$$\tilde{J}(x_\ell) = \sum_{k=\ell}^{\ell+m-1} g(x_k, \mu(x_k)) + \hat{J}(x_{\ell+m})$$

- ℓ-step lookahead: optimizing over ℓ controls $u_0, u_1, \ldots, u_{\ell-1}$:

$$\tilde{\mu}(x_0) \in \arg \min_{u_0 \in U(x_0)} \left(g(x_0, u_0) + \min_{u_k,\, k=1,\ldots,\ell-1} \Big(\sum_{k=1}^{\ell-1} g(x_k, u_k) + \tilde{J}(x_\ell) \Big) \right).$$

- Simplified minimization: construct a subset $\bar{U}(x) \subset U(x)$, and compute $\tilde{\mu}(x)$ via

$$\tilde{\mu}(x) \in \arg \min_{u \in \bar{U}(x)} \left\{ g(x, u) + \tilde{J}\big(f(x, u)\big) \right\}.$$

- All the ideas discussed thus far apply to finite horizon problems!

MPC as Approximation in Value Space

ON-LINE PLAY

$$\min_{\{u_k\}_{k=0}^{\ell-1}} \sum_{k=0}^{\ell+m-1} g(x_k, u_k) + G(x_{\ell+m}) \qquad \text{terminal cost}$$

$$\begin{aligned} \text{s. t.} \quad & x_{k+1} = f(x_k, u_k),\ k = 0, \dots, \ell+m-1, \\ & x_k \in C,\ u_k \in U(x_k),\ \ k = 0, \dots, \ell+m-1, \\ & u_k = \mu(x_k),\ \ k = \ell, \dots, \ell+m-1, \qquad \text{base policy} \\ & x_{\ell+m} \in C_{\ell+m} \qquad \text{terminal constraint} \\ & x_0 = x. \end{aligned}$$

OFF-LINE TRAINING

- Offline training: The cost functions J_μ of some base policy can be computed as closed-form expressions and used as G.
- Online play: The minimization problems are cast as optimization problems that can be solved efficiently.
- These favorable characteristics of MPC may not be present in other context. But we have remedies.

MPC as Approximation in Value Space

ON-LINE PLAY

$$\min_{\{u_k\}_{k=0}^{\ell-1}} \sum_{k=0}^{\ell+m-1} g(x_k, u_k) + G(x_{\ell+m}) \qquad \text{terminal cost}$$

$$\begin{aligned} \text{s. t.} \quad & x_{k+1} = f(x_k, u_k),\ k = 0, ..., \ell + m - 1, \\ & x_k \in C,\ u_k \in U(x_k),\ \ k = 0, ..., \ell + m - 1, \\ & u_k = \mu(x_k),\ \ k = \ell, ..., \ell + m - 1, \quad \text{base policy} \\ & x_{\ell+m} \in C_{\ell+m} \quad \text{terminal constraint} \\ & x_0 = x. \end{aligned}$$

OFF-LINE TRAINING

- Offline training: The cost functions J_μ of some base policy can be computed as closed-form expressions and used as G.
- Online play: The minimization problems are cast as optimization problems that can be solved efficiently.
- These favorable characteristics of MPC may not be present in other context. But we have remedies.

MPC as Approximation in Value Space

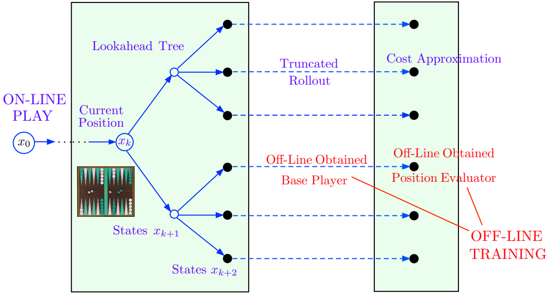
ON-LINE PLAY

$$\min_{\{u_k\}_{k=0}^{\ell-1}} \sum_{k=0}^{\ell+m-1} g(x_k, u_k) + G(x_{\ell+m}) \qquad \text{terminal cost}$$

$$\begin{aligned} \text{s. t.} \quad & x_{k+1} = f(x_k, u_k),\ k = 0, ..., \ell + m - 1, \\ & x_k \in C,\ u_k \in U(x_k),\ \ k = 0, ..., \ell + m - 1, \\ & u_k = \mu(x_k),\ \ k = \ell, ..., \ell + m - 1, \qquad \text{base policy} \\ & x_{\ell+m} \in C_{\ell+m} \qquad \text{terminal constraint} \\ & x_0 = x. \end{aligned}$$

OFF-LINE TRAINING

- Offline training: The cost functions J_μ of some base policy can be computed as closed-form expressions and used as G.
- Online play: The minimization problems are cast as optimization problems that can be solved efficiently.
- These favorable characteristics of MPC may not be present in other context. But we have remedies.

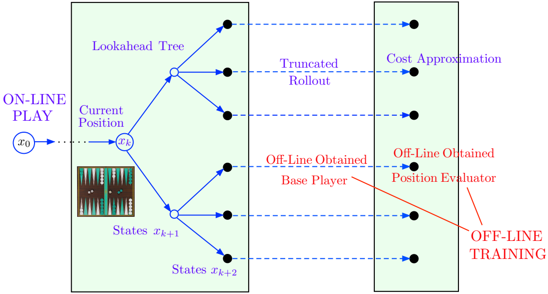


- Training a neural network to represent the function $\hat{J}$
- Using real-time simulation to make up the imperfect $\hat{J}$: truncated rollout to collect sample trajectory $x_{k+2}, \mu(x_{k+2}), x_{k+3}, \ldots, x_{k+2+m}$, and the effective $\tilde{J}$ is

$$\tilde{J}(x_{k+2}) = \sum_{i=k+2}^{k+2+m-1} g(x_i, \mu(x_i)) + \hat{J}(x_{k+2+m})$$

- The lookahead tree is constructed for the minization computation.

Remedies from TD-Gammon

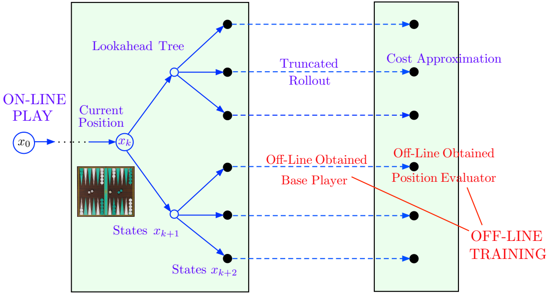


- Training a neural network to represent the function $\hat{J}$
- Using real-time simulation to make up the imperfect $\hat{J}$: truncated rollout to collect sample trajectory $x_{k+2}, \mu(x_{k+2}), x_{k+3}, \ldots, x_{k+2+m}$, and the effective $\tilde{J}$ is

$$\tilde{J}(x_{k+2}) = \sum_{i=k+2}^{k+2+m-1} g(x_i, \mu(x_i)) + \hat{J}(x_{k+2+m})$$

- The lookahead tree is constructed for the minization computation.

Remedies from TD-Gammon



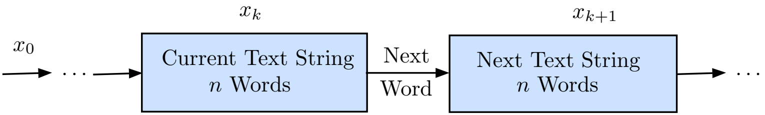
- Training a neural network to represent the function $\hat{J}$
- Using real-time simulation to make up the imperfect $\hat{J}$: truncated rollout to collect sample trajectory $x_{k+2}, \mu(x_{k+2}), x_{k+3}, \ldots, x_{k+2+m}$, and the effective $\tilde{J}$ is

$$\tilde{J}(x_{k+2}) = \sum_{i=k+2}^{k+2+m-1} g(x_i, \mu(x_i)) + \hat{J}(x_{k+2+m})$$

- The lookahead tree is constructed for the minization computation.

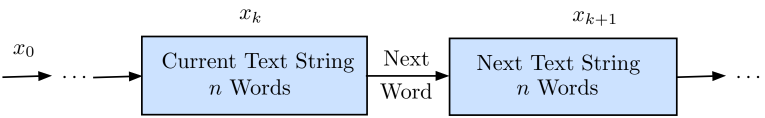
Outline

The n-Gram Model of Next Word Generation



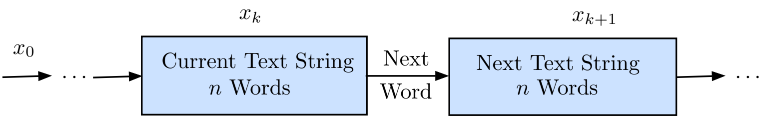
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

The n-Gram Model of Next Word Generation



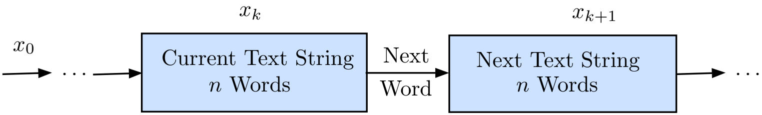
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

The n-Gram Model of Next Word Generation



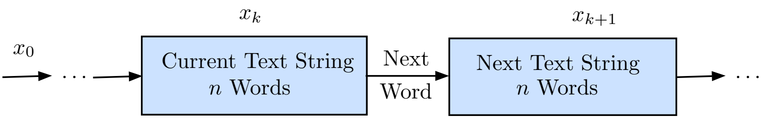
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

The n-Gram Model of Next Word Generation



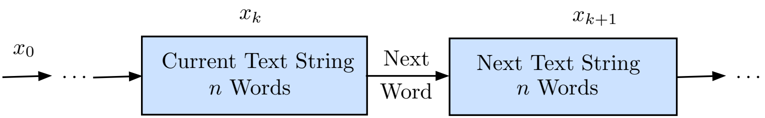
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

The n-Gram Model of Next Word Generation



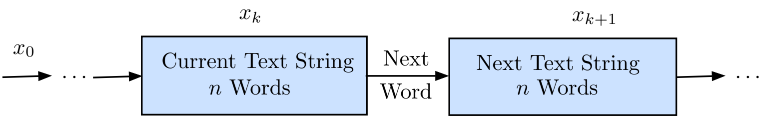
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

The n-Gram Model of Next Word Generation



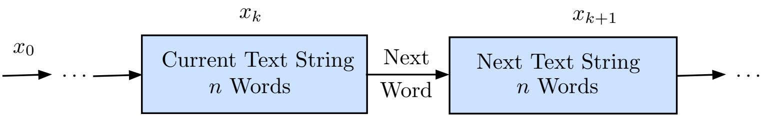
- One word added to the front and one word deleted from the back
- The n-gram provides transition probabilities $p(x_{k+1} \mid x_k)$ to which we have access
- $p(x_{k+1} \mid x_k)$ is a suggested local measure of desirability for x_{k+1} to follow x_k
- We have freedom to select the next word according 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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

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
- The constant n is also known as the size of the context window, and the constant N is the generated sequence length

An Optimization Problem: Most Likely Sequence Selection Poiicy



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

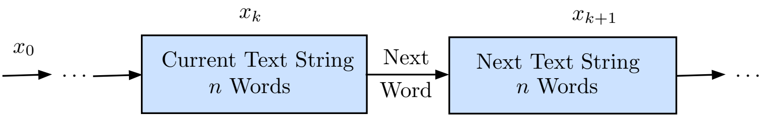
$$\text{Prob}(x_1, x_2, \ldots, x_N \mid x_0)$$

  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.

An Optimization Problem: Most Likely Sequence Selection Poiicy



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

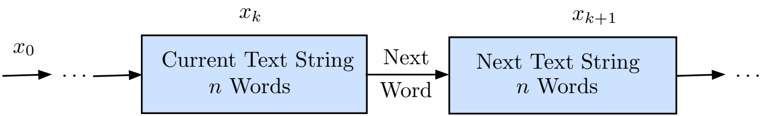
$$\text{Prob}(x_1, x_2, \ldots, x_N \mid x_0)$$

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.

An Optimization Problem: Most Likely Sequence Selection Poiicy



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

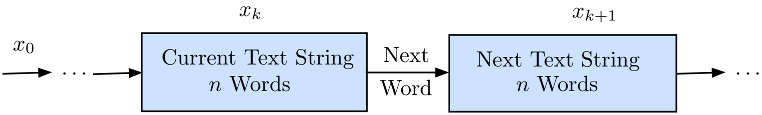
$$\text{Prob}(x_1, x_2, \ldots, x_N \mid x_0)$$

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.

Dynamic Programming Formulation of the Problem



- The control constraint sets $U(x)$: the set of all possible words U, known as the vocabulary, independent of x
- The state space X: the n-fold product of U, i.e., $X = U^n$
- The system dynamics f: Given a text string (state) x_k and a word (control) u_k, the new text string (next state) x_{k+1} is obtained by

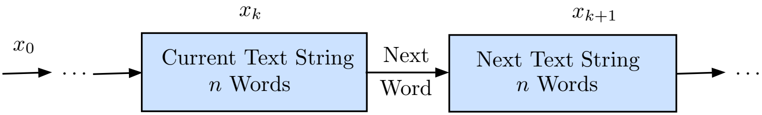
  adding u_k to the front end of x_k, and deleting the last word at the back end of x_k

- The stage cost g: The cost of applying u_k at x_k is given by

$$g(x_k, u_k) = -\log p(x_{k+1} \mid x_k),$$

  where $x_{k+1} = f(x_k, u_k)$. For convenience, we still work with the probabilities directly
- This is a finite-horizon problem: selecting N controls

Dynamic Programming Formulation of the Problem



- The control constraint sets $U(x)$: the set of all possible words U, known as the vocabulary, independent of x
- The state space X: the n-fold product of U, i.e., $X = U^n$
- The system dynamics f: Given a text string (state) x_k and a word (control) u_k, the new text string (next state) x_{k+1} is obtained by

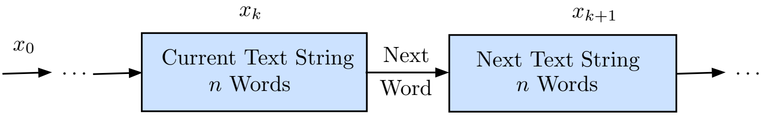
  adding u_k to the front end of x_k, and deleting the last word at the back end of x_k

- The stage cost g: The cost of applying u_k at x_k is given by

$$g(x_k, u_k) = -\log p(x_{k+1} \mid x_k),$$

  where $x_{k+1} = f(x_k, u_k)$. For convenience, we still work with the probabilities directly
- This is a finite-horizon problem: selecting N controls

Dynamic Programming Formulation of the Problem



- The control constraint sets $U(x)$: the set of all possible words U, known as the vocabulary, independent of x
- The state space X: the n-fold product of U, i.e., $X = U^n$
- The system dynamics f: Given a text string (state) x_k and a word (control) u_k, the new text string (next state) x_{k+1} is obtained by

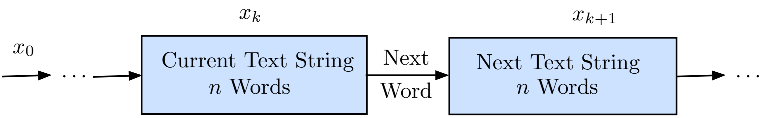
  adding u_k to the front end of x_k, and deleting the last word at the back end of x_k

- The stage cost g: The cost of applying u_k at x_k is given by

$$g(x_k, u_k) = -\log p(x_{k+1} \mid x_k),$$

  where $x_{k+1} = f(x_k, u_k)$. For convenience, we still work with the probabilities directly
- This is a finite-horizon problem: selecting N controls

Dynamic Programming Formulation of the Problem



- The control constraint sets $U(x)$: the set of all possible words U, known as the vocabulary, independent of x
- The state space X: the n-fold product of U, i.e., $X = U^n$
- The system dynamics f: Given a text string (state) x_k and a word (control) u_k, the new text string (next state) x_{k+1} is obtained by

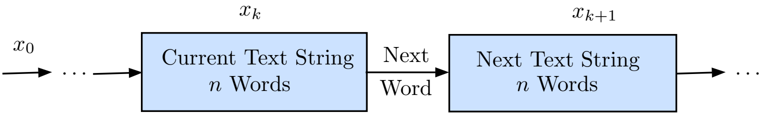
  adding u_k to the front end of x_k, and deleting the last word at the back end of x_k

- The stage cost g: The cost of applying u_k at x_k is given by

$$g(x_k, u_k) = -\log p(x_{k+1} \mid x_k),$$

  where $x_{k+1} = f(x_k, u_k)$. For convenience, we still work with the probabilities directly
- This is a finite-horizon problem: selecting N controls

Rollout Policy Based on the Greedy Policy

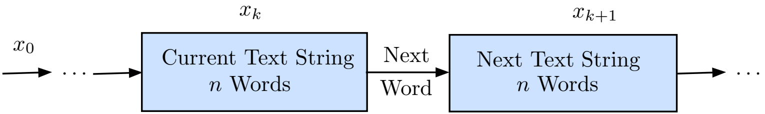


- The optimal selection policy: Intractable to compute when U and/or n are large.
- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects u_k that maximizes the greedy Q-factor $Q(x_k, u_k)$; i.e. the probability of the sequence

$$\text{Prob}\big(f(x_k, u_k), \text{Greedy sequence starting from } f(x_k, u_k) \mid x_k\big)$$

- In this case, the function $\tilde{J}$ is J_μ, with μ being the greedy selection policy
- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.

Rollout Policy Based on the Greedy Policy

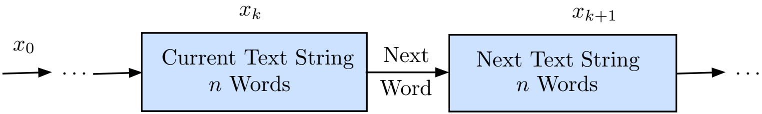


- The optimal selection policy: Intractable to compute when U and/or n are large.
- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects u_k that maximizes the greedy Q-factor $Q(x_k, u_k)$; i.e. the probability of the sequence

$$\text{Prob}(f(x_k, u_k), \text{Greedy sequence starting from } f(x_k, u_k) \mid x_k)$$

- In this case, the function $\tilde{J}$ is J_μ, with μ being the greedy selection policy
- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.

Rollout Policy Based on the Greedy Policy

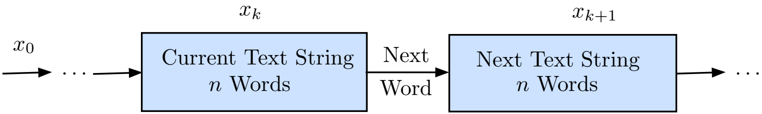


- The optimal selection policy: Intractable to compute when U and/or n are large.
- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects u_k that maximizes the greedy Q-factor $Q(x_k, u_k)$; i.e. the probability of the sequence

$$\text{Prob}\big(f(x_k, u_k), \text{Greedy sequence starting from } f(x_k, u_k) \mid x_k\big)$$

- In this case, the function $\tilde{J}$ is J_μ, with μ being the greedy selection policy
- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.

Rollout Policy Based on the Greedy Policy

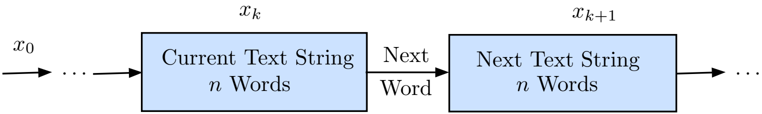


- The optimal selection policy: Intractable to compute when U and/or n are large.
- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects u_k that maximizes the greedy Q-factor $Q(x_k, u_k)$; i.e. the probability of the sequence

$$\text{Prob}(f(x_k, u_k), \text{Greedy sequence starting from } f(x_k, u_k) \mid x_k)$$

- In this case, the function $\tilde{J}$ is J_μ, with μ being the greedy selection policy
- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.

Rollout Policy Based on the Greedy Policy

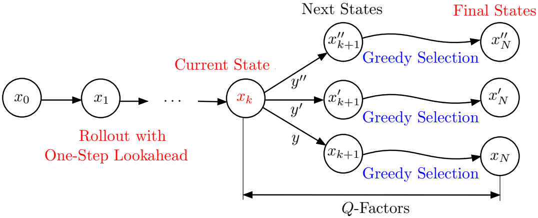


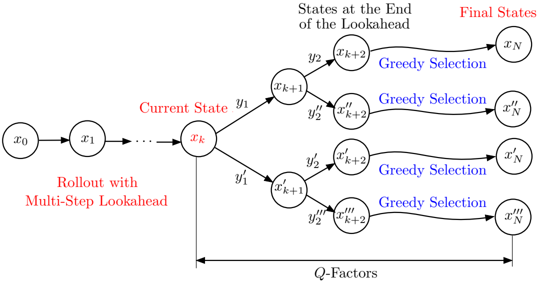
- The optimal selection policy: Intractable to compute when U and/or n are large.
- The greedy selection policy: Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} \mid x_k)$.
- The rollout selection policy that uses the greedy as base policy: At x_k, it selects u_k that maximizes the greedy Q-factor $Q(x_k, u_k)$; i.e. the probability of the sequence

$$\text{Prob}\big(f(x_k, u_k), \text{Greedy sequence starting from } f(x_k, u_k) \mid x_k\big)$$

- In this case, the function $\tilde{J}$ is J_μ, with μ being the greedy selection policy
- Variants of rollout: Multistep lookahead, truncated, simplified, and their combinations.

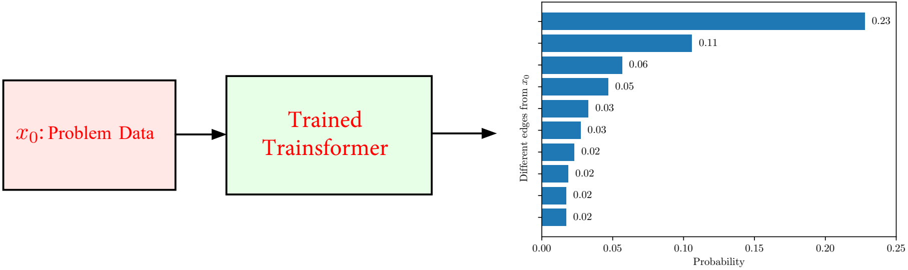
One-Step and Multistep Rollout Selection Policies





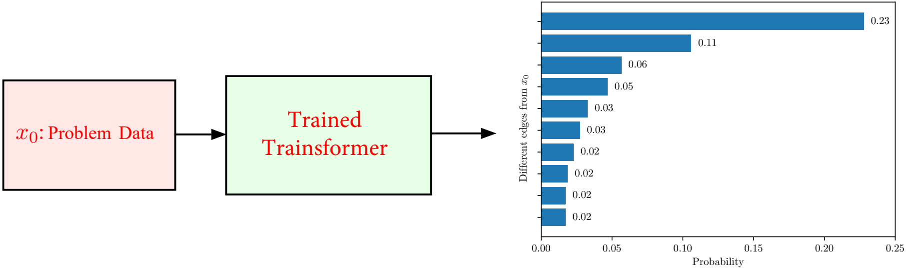
There are also truncated and simplified variants, etc

Most Likely Word Sequence from a GPT



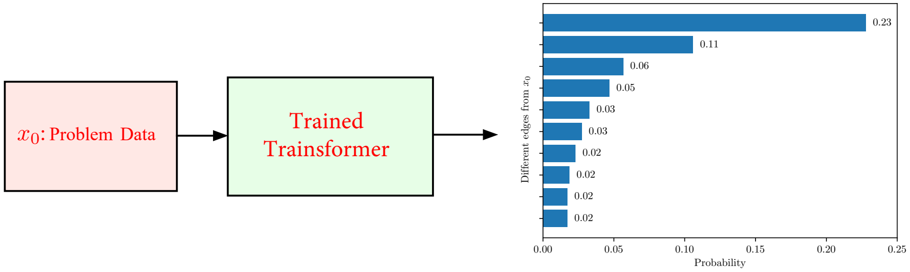
- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- 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)

Most Likely Word Sequence from a GPT

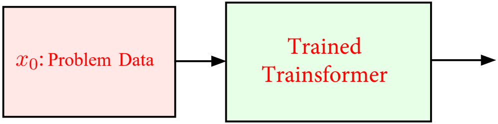


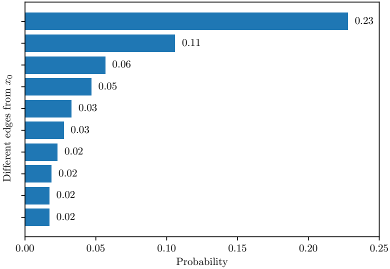
- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- 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)

Most Likely Word Sequence from a GPT



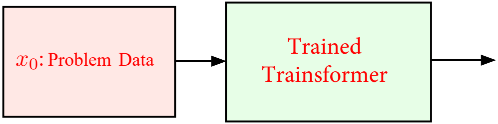
- We generated most likely sequences, using a fine-tuned GPT, which defines an *n*-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different *n*-grams is 50258^{1024}, enormous! Intractable via 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)

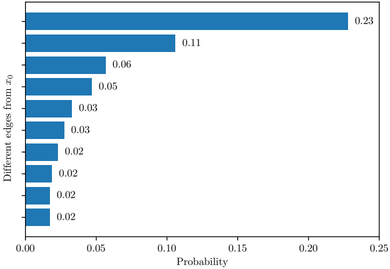




- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- 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)

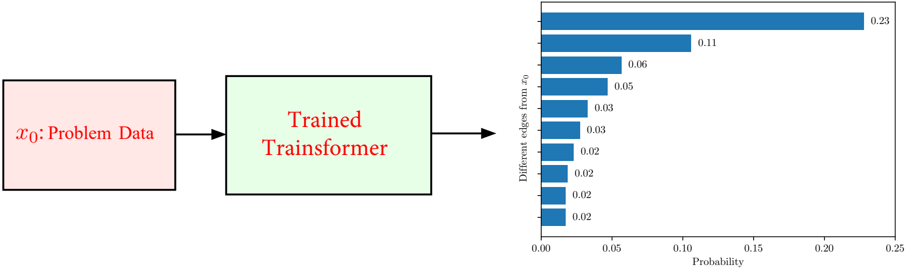
Most Likely Word Sequence from a GPT





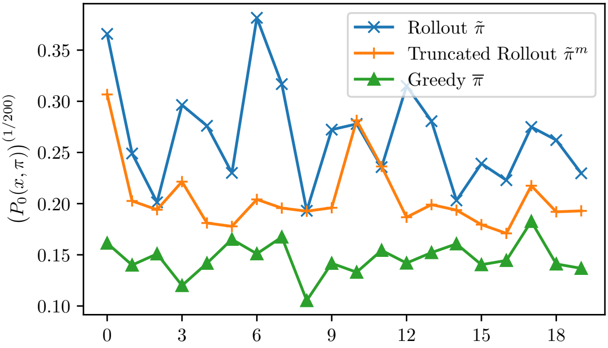
- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- 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)

Most Likely Word Sequence from a GPT



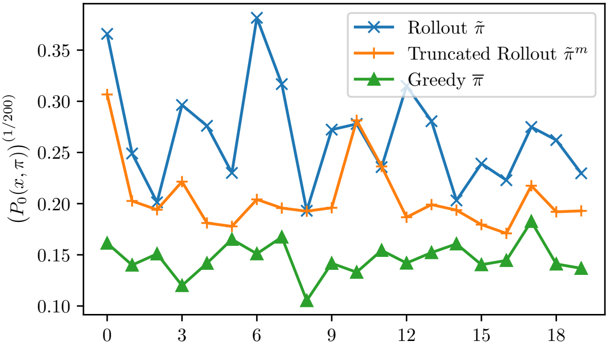
- We generated most likely sequences, using a fine-tuned GPT, which defines an n-gram and its associated transition probabilities. We used $N = 200$ and $n = 1024$.
- The transition probabilities are generated by the GPT
- The number of different n-grams is 50258^{1024}, enormous! Intractable via DP
- 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



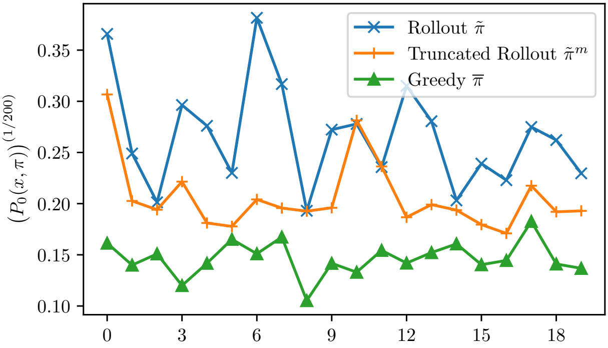
- We applied two simplification techniques:
  - Computing only 10 Q-factors corresponding to top ten most likely next words: simplified rollout with one-step lookahead
  - In addition, truncating the simulation after 10 steps: *m*-step truncated rollout
- General observations from the experiments:
  - 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

Performance of Simplified Rollout



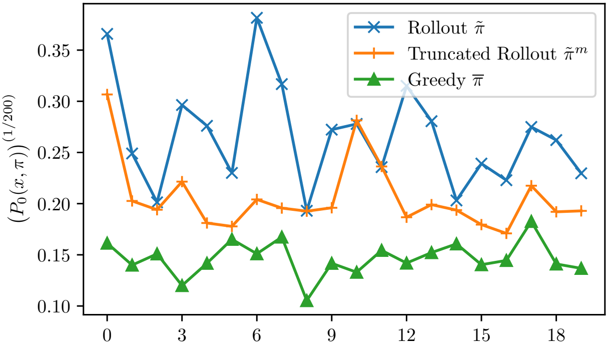
- We applied two simplification techniques:
  - Computing only 10 Q-factors corresponding to top ten most likely next words: simplified rollout with one-step lookahead
  - In addition, truncating the simulation after 10 steps: *m*-step truncated rollout
- General observations from the experiments:
  - 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

Performance of Simplified Rollout



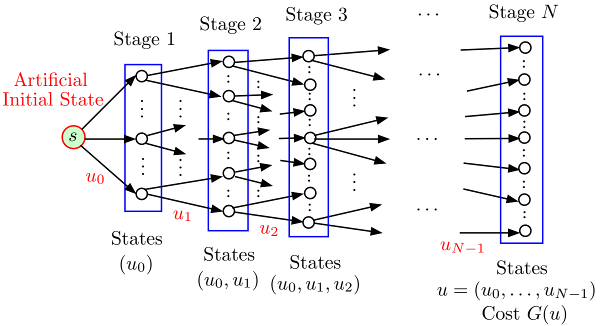
- We applied two simplification techniques:
  - Computing only 10 Q-factors corresponding to top ten most likely next words: simplified rollout with one-step lookahead
  - In addition, truncating the simulation after 10 steps: *m*-step truncated rollout
- General observations from the experiments:
  - 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

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

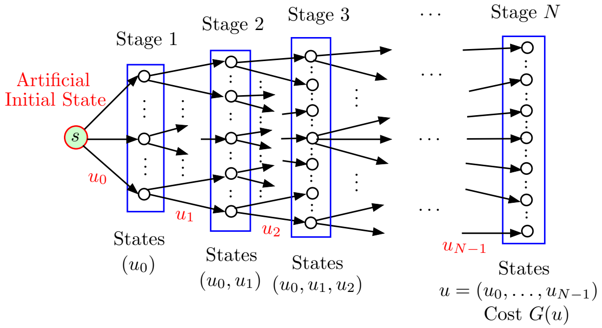
Outline



Minimize $G(u)$ subject to $u \in U$

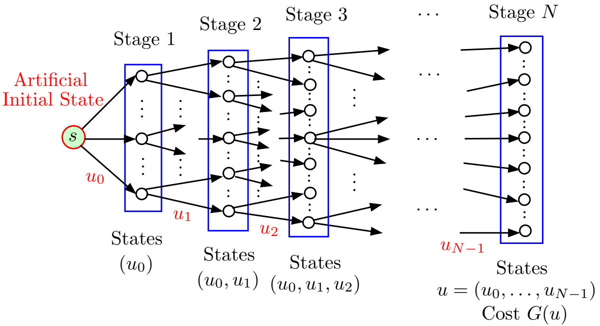
- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- The system dynamics is $f(x_k, u_k) = (u_0, \ldots, u_{k-1}, u_k)$, where $x_k = (u_0, \ldots, u_{k-1})$.
- Only the state and control pairs (x_{N-1}, u_N) has the cost $g(x_{N-1}, u_N) = G(u)$; all other costs are 0

Modeling General Discrete Optimization via DP



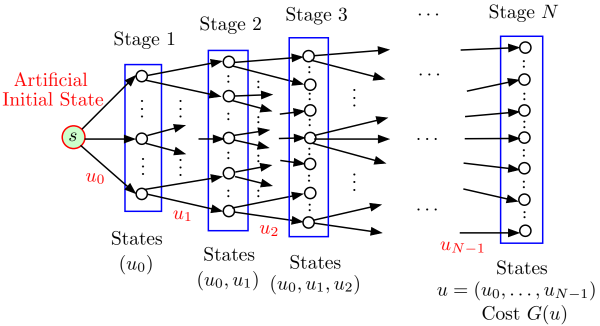
Minimize $G(u)$ subject to $u \in U$

- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- The system dynamics is $f(x_k, u_k) = (u_0, \ldots, u_{k-1}, u_k)$, where $x_k = (u_0, \ldots, u_{k-1})$.
- Only the state and control pairs (x_{N-1}, u_N) has the cost $g(x_{N-1}, u_N) = G(u)$; all other costs are 0



Minimize $G(u)$ subject to $u \in U$

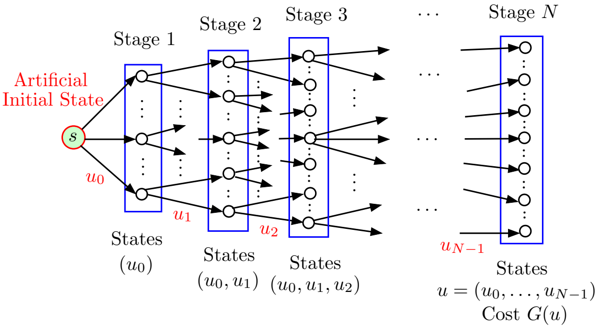
- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- The system dynamics is $f(x_k, u_k) = (u_0, \ldots, u_{k-1}, u_k)$, where $x_k = (u_0, \ldots, u_{k-1})$.
- Only the state and control pairs (x_{N-1}, u_N) has the cost $g(x_{N-1}, u_N) = G(u)$; all other costs are 0



Minimize $G(u)$ subject to $u \in U$

- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- The system dynamics is $f(x_k, u_k) = (u_0, \ldots, u_{k-1}, u_k)$, where $x_k = (u_0, \ldots, u_{k-1})$.
- Only the state and control pairs (x_{N-1}, u_N) has the cost $g(x_{N-1}, u_N) = G(u)$; all other costs are 0

Modeling General Discrete Optimization via DP



Minimize $G(u)$ subject to $u \in U$

- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- The system dynamics is $f(x_k, u_k) = (u_0, \ldots, u_{k-1}, u_k)$, where $x_k = (u_0, \ldots, u_{k-1})$.
- Only the state and control pairs (x_{N-1}, u_N) has the cost $g(x_{N-1}, u_N) = G(u)$; all other costs are 0

DP solution to the discrete optimization problem

- Start with

$$J_N^*(x_N) = G(x_N) = G(u_0, \dots, u_{N-1}) \quad \text{for all } x_N \in U$$

- For $k = 0, \dots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

where $U_k(x_k)$ need to be suitably defined.

- Construct the optimal solution $(u_0^*, \dots, u_{N-1}^*)$ by forward calculation

$$u_k^* \in \arg \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \dots, N-1$, set

$$\tilde{\mu}(x_k) \in \arg \min_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

DP and Approximation in Value Space

DP solution to the discrete optimization problem

- Start with

$$J_N^*(x_N) = G(x_N) = G(u_0, \ldots, u_{N-1}) \quad \text{for all } x_N \in U$$

- For $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

where $U_k(x_k)$ need to be suitably defined.

- Construct the optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ by forward calculation

$$u_k^* \in \arg\min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \ldots, N-1$, set

$$\tilde{\mu}(x_k) \in \arg\min_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

DP and Approximation in Value Space

DP solution to the discrete optimization problem

- Start with

$$J_N^*(x_N) = G(x_N) = G(u_0, \ldots, u_{N-1}) \quad \text{for all } x_N \in U$$

- For $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

where $U_k(x_k)$ need to be suitably defined.

- Construct the optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ by forward calculation

$$u_k^* \in \arg\min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \ldots, N-1$, set

$$\tilde{\mu}(x_k) \in \arg\min_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

DP and Approximation in Value Space

DP solution to the discrete optimization problem

- Start with

$$J_N^*(x_N) = G(x_N) = G(u_0, \ldots, u_{N-1}) \quad \text{for all } x_N \in U$$

- For $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

  where $U_k(x_k)$ need to be suitably defined.
- Construct the optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ by forward calculation

$$u_k^* \in \arg \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \ldots, N-1$, set

$$\tilde{\mu}(x_k) \in \arg \min_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

DP and Approximation in Value Space

DP solution to the discrete optimization problem

- Start with

$$J_N^*(x_N) = G(x_N) = G(u_0, \dots, u_{N-1}) \quad \text{for all } x_N \in U$$

- For $k = 0, \dots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

  where $U_k(x_k)$ need to be suitably defined.
- Construct the optimal solution $(u_0^*, \dots, u_{N-1}^*)$ by forward calculation

$$u_k^* \in \arg\min_{u_k \in U(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \dots, N-1$, set

$$\tilde{\mu}(x_k) \in \arg\min_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

Multiple Object Tracking

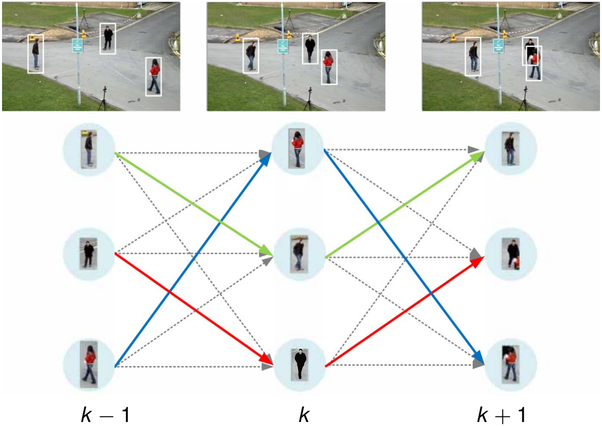
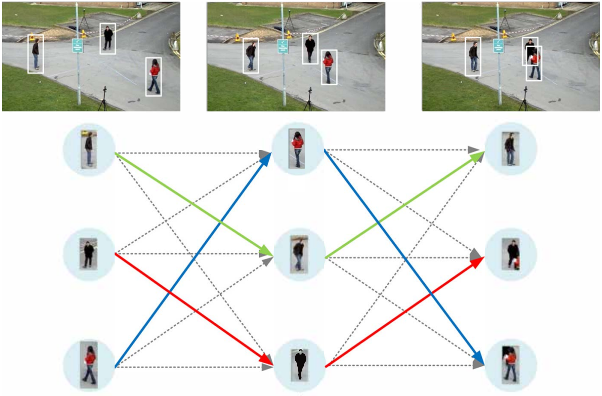


Figure source: Chumachenko et. al., Object Detection and Tracking

- Multiple object tracking (MOT) aims to match the same objects over various frames
- Nontrivial: occlusion, changes in object appearance, and real-time computation constraint
- Important problem with many applications: traffic monitoring, robotics, consumer analytics, augmented and virtual realities ...

Multiple Object Tracking



$k-1$ k $k+1$

Figure source: Chumachenko et. al., Object Detection and Tracking

- Multiple object tracking (MOT) aims to match the same objects over various frames
- Nontrivial: occlusion, changes in object appearance, and real-time computation constraint
- Important problem with many applications: traffic monitoring, robotics, consumer analytics, augmented and virtual realities ...

Multiple Object Tracking

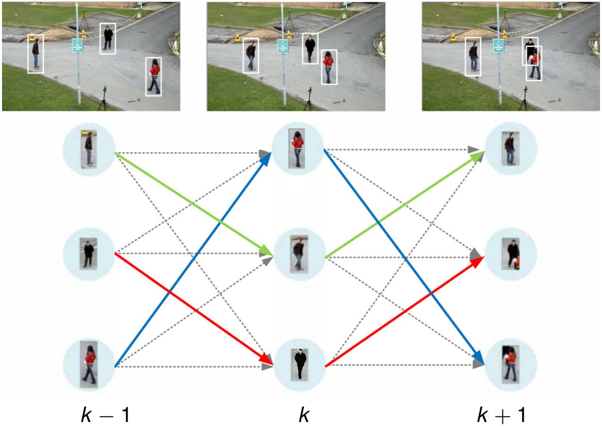
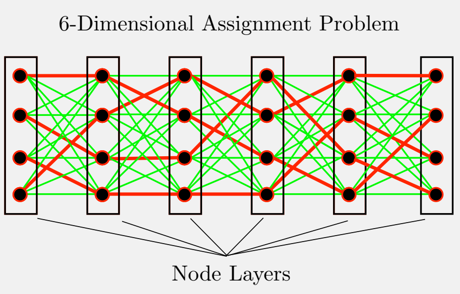


Figure source: Chumachenko et. al., Object Detection and Tracking

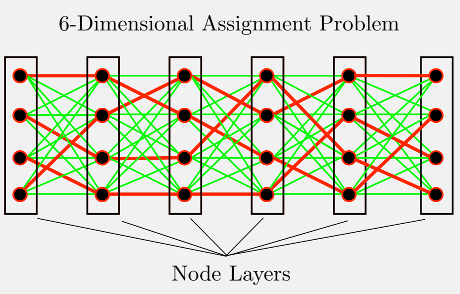
- Multiple object tracking (MOT) aims to match the same objects over various frames
- Nontrivial: occlusion, changes in object appearance, and real-time computation constraint
- Important problem with many applications: traffic monitoring, robotics, consumer analytics, augmented and virtual realities ...

Multidimensional Assignment Problem



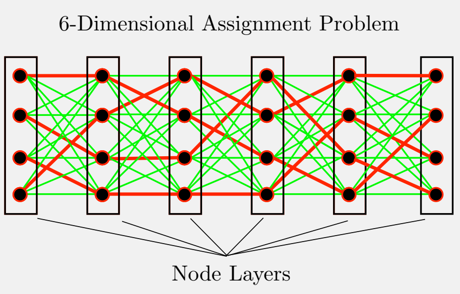
- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \ldots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized

Multidimensional Assignment Problem

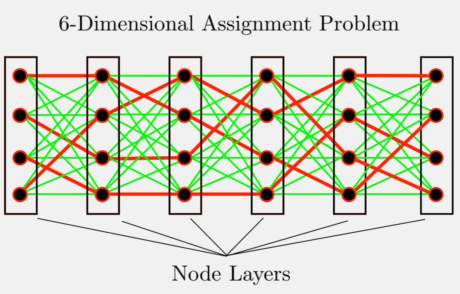


- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \ldots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized

Multidimensional Assignment Problem

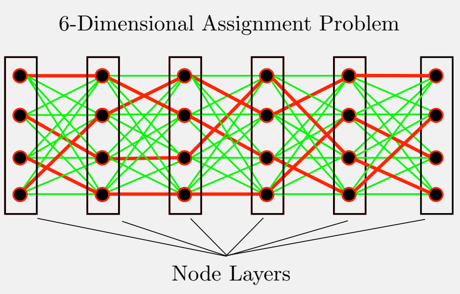


- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \ldots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized



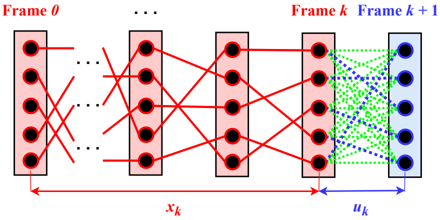
- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \dots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized

Multidimensional Assignment Problem



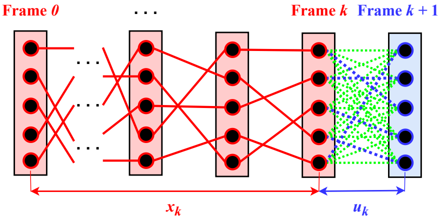
- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \ldots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized

DP Formulation for MOT

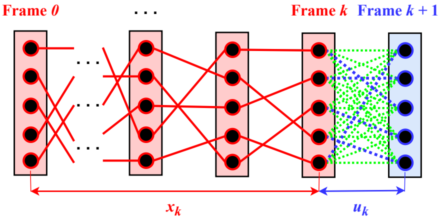


- The state $x_k = (u_0, u_1, \ldots, u_{k-1})$ defines a set of tracks, referred to as the given tracks.
- At each time, we select u_k in order to match the objects in the target frame to the given tracks
- Each u_k is a matching between the tracks and the objects in the target frame

DP Formulation for MOT

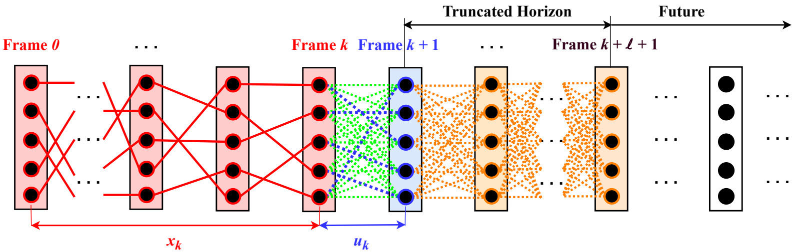


- The state $x_k = (u_0, u_1, \ldots, u_{k-1})$ defines a set of tracks, referred to as the given tracks.
- At each time, we select u_k in order to match the objects in the target frame to the given tracks
- Each u_k is a matching between the tracks and the objects in the target frame



- The state $x_k = (u_0, u_1, \ldots, u_{k-1})$ defines a set of tracks, referred to as the given tracks.
- At each time, we select u_k in order to match the objects in the target frame to the given tracks
- Each u_k is a matching between the tracks and the objects in the target frame

Approximation in Value Space for MOT

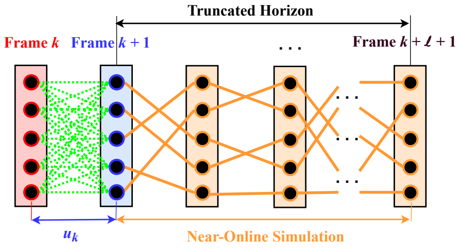


- First process a few frames beyond target frame defined by the truncated horizon
- It then applies a base policy to solve the MOT starting from the target frame, which we call near-online simulation
- The function $\tilde{J}_{k+1}(x_k, u_k)$ is given by the sum of similarity scores $c^{ij}_{k+1}(x_k)$:

$$\tilde{J}_{k+1}(x_k, u_k) = \sum_{(i,j)\in u_k} c^{ij}_{k+1}(x_k),$$

  where each $(i, j) \in u_k$ is an arc from the matching specified by control u_k
- The control selection $\tilde{\mu}(x_k) \in \arg\max_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k)$ becomes solving a bipartite matching problem

Approximation in Value Space for MOT



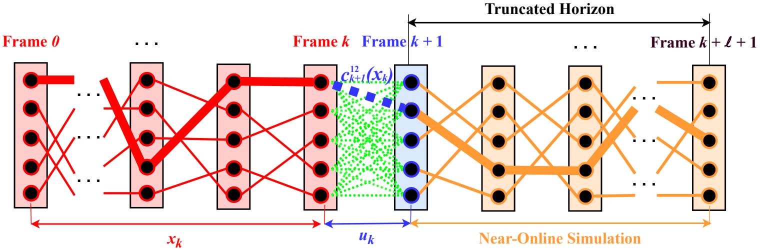
- First process a few frames beyond target frame defined by the truncated horizon
- It then applies a base policy to solve the MOT starting from the target frame, which we call near-online simulation
- The function $\tilde{J}_{k+1}(x_k, u_k)$ is given by the sum of similarity scores $c^{ij}_{k+1}(x_k)$:

$$\tilde{J}_{k+1}(x_k, u_k) = \sum_{(i,j)\in u_k} c^{ij}_{k+1}(x_k),$$

where each $(i, j) \in u_k$ is an arc from the matching specified by control u_k

- The control selection $\tilde{\mu}(x_k) \in \arg\max_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k)$ becomes solving a bipartite matching problem

Approximation in Value Space for MOT



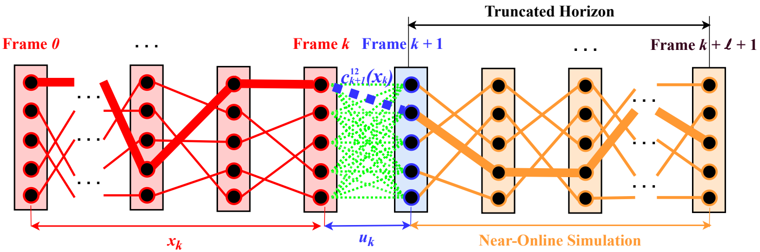
- First process a few frames beyond target frame defined by the truncated horizon
- It then applies a base policy to solve the MOT starting from the target frame, which we call near-online simulation
- The function $\tilde{J}_{k+1}(x_k, u_k)$ is given by the sum of similarity scores $c^{ij}_{k+1}(x_k)$:

$$\tilde{J}_{k+1}(x_k, u_k) = \sum_{(i,j)\in u_k} c^{ij}_{k+1}(x_k),$$

where each $(i, j) \in u_k$ is an arc from the matching specified by control u_k

- The control selection $\tilde{\mu}(x_k) \in \arg\max_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k)$ becomes solving a bipartite matching problem

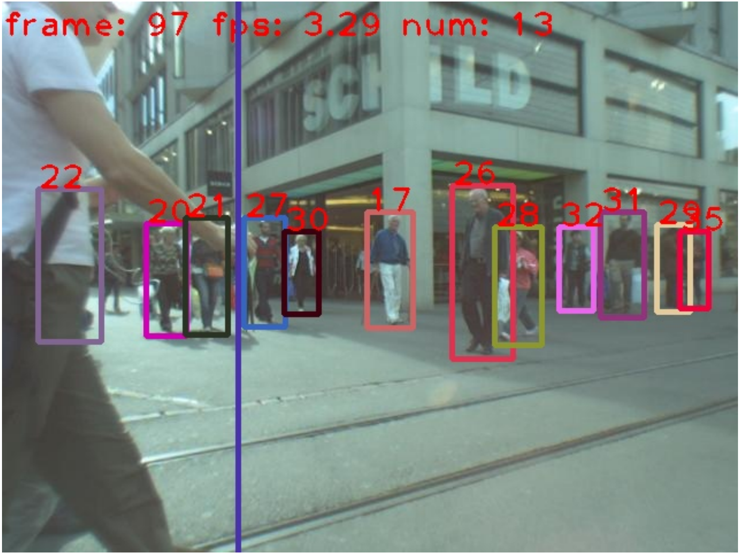
Approximation in Value Space for MOT



- First process a few frames beyond target frame defined by the truncated horizon
- It then applies a base policy to solve the MOT starting from the target frame, which we call near-online simulation
- The function $\tilde{J}_{k+1}(x_k, u_k)$ is given by the sum of similarity scores $c^{ij}_{k+1}(x_k)$:

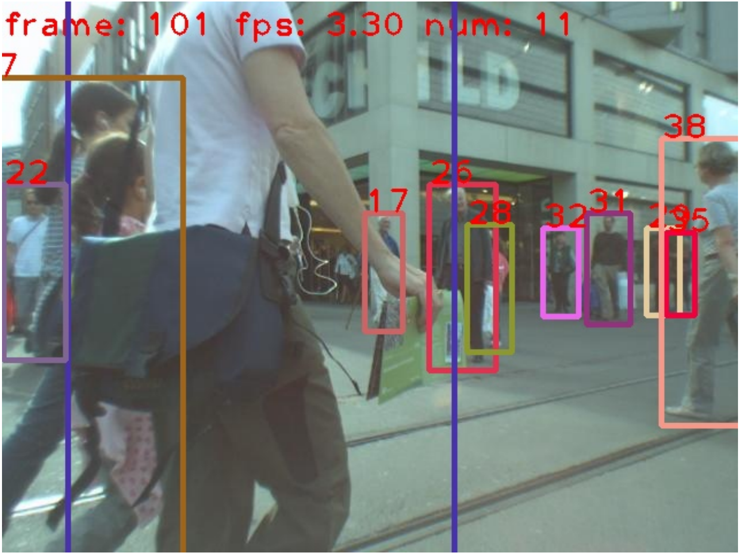
$$\tilde{J}_{k+1}(x_k, u_k) = \sum_{(i,j)\in u_k} c^{ij}_{k+1}(x_k),$$

  where each $(i, j) \in u_k$ is an arc from the matching specified by control u_k
- The control selection $\tilde{\mu}(x_k) \in \arg\max_{u_k \in U(x_k)} \tilde{J}_{k+1}(x_k, u_k)$ becomes solving a bipartite matching problem


frame: 97 fps: 3.29 num: 13
22
20
21
27
30
17
26
28
32
31
29
5

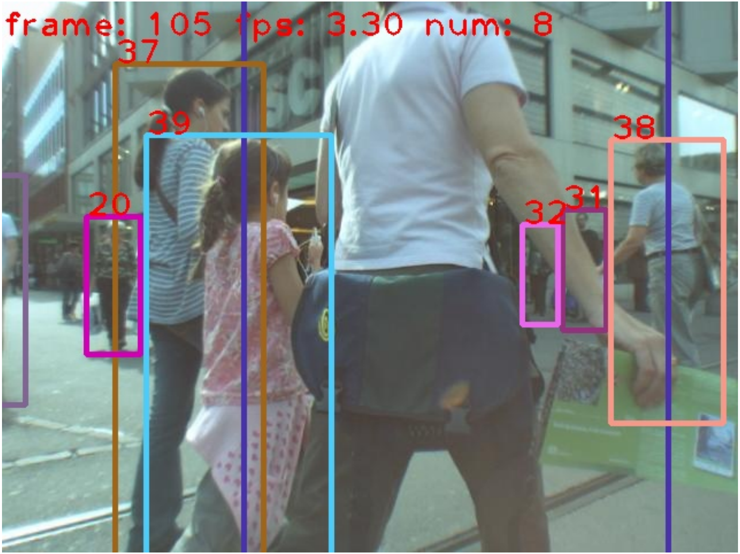
MOT Example: Base Policy

MOT Example: Base Policy



MOT Example: Base Policy

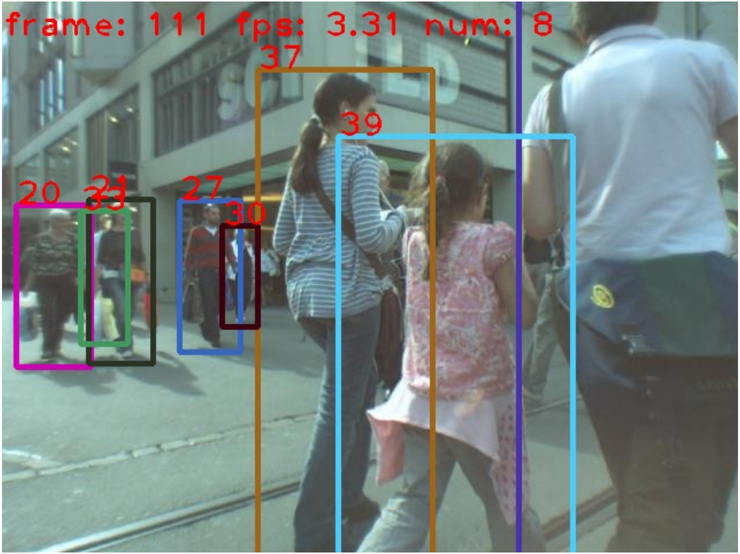
MOT Example: Base Policy



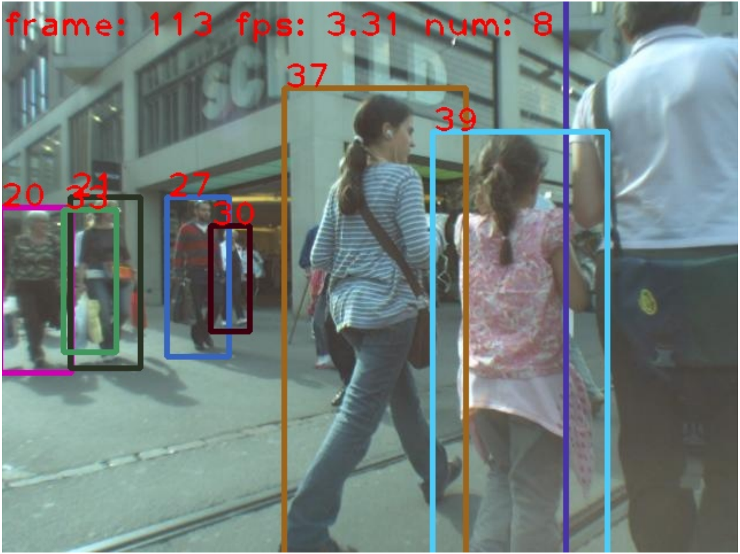
MOT Example: Base Policy

MOT Example: Base Policy

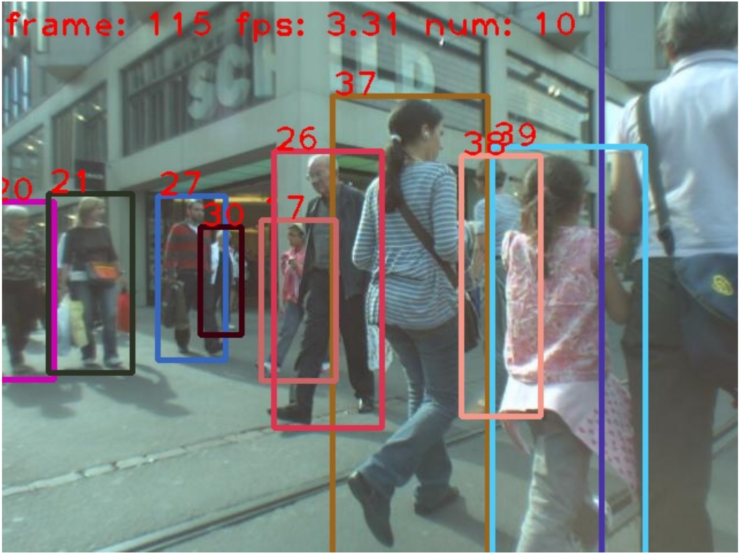
MOT Example: Base Policy



MOT Example: Base Policy

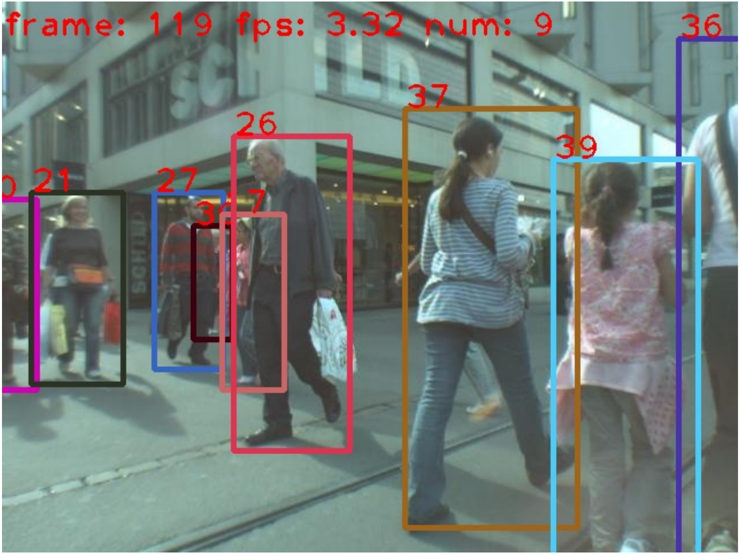


MOT Example: Base Policy

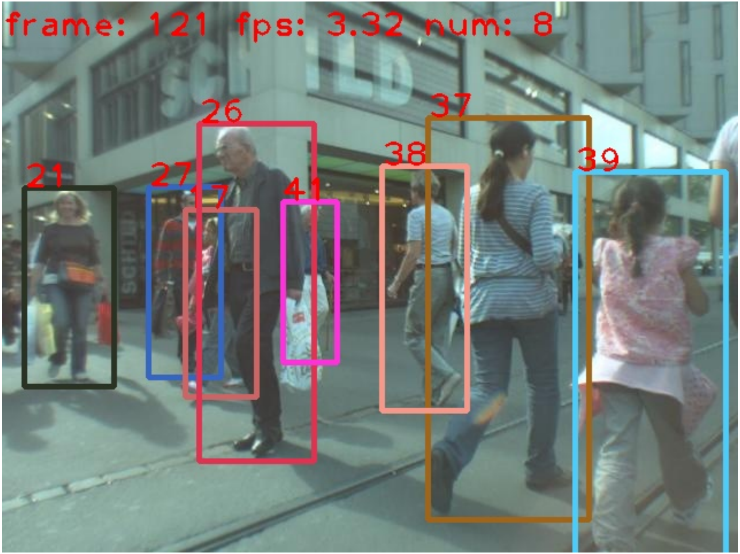


MOT Example: Base Policy

MOT Example: Base Policy



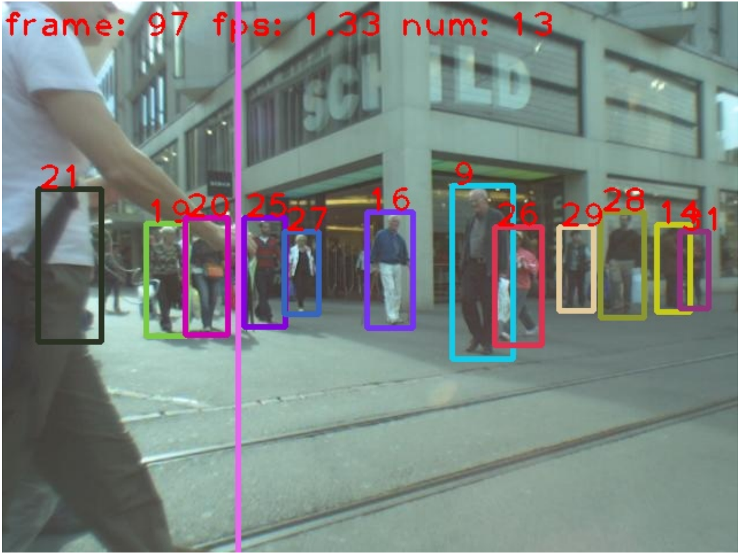
MOT Example: Base Policy



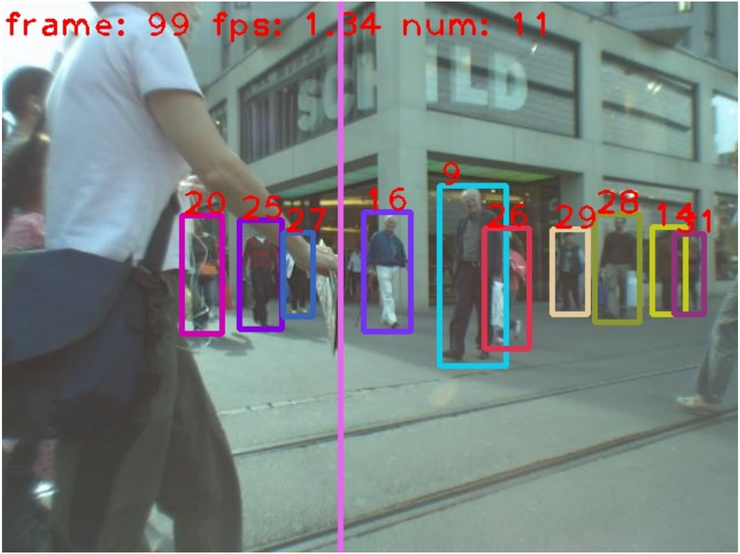
MOT Example: Base Policy

MOT Example: Base Policy

MOT Example: Approximation in Value Space



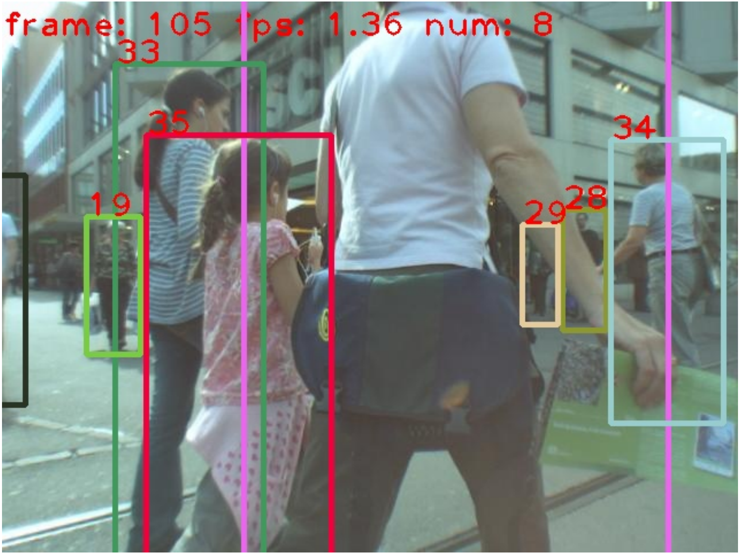
MOT Example: Approximation in Value Space




frame: 101 fps: 1.34 num: 11
3
21
16
9
26
29
28
34

MOT Example: Approximation in Value Space

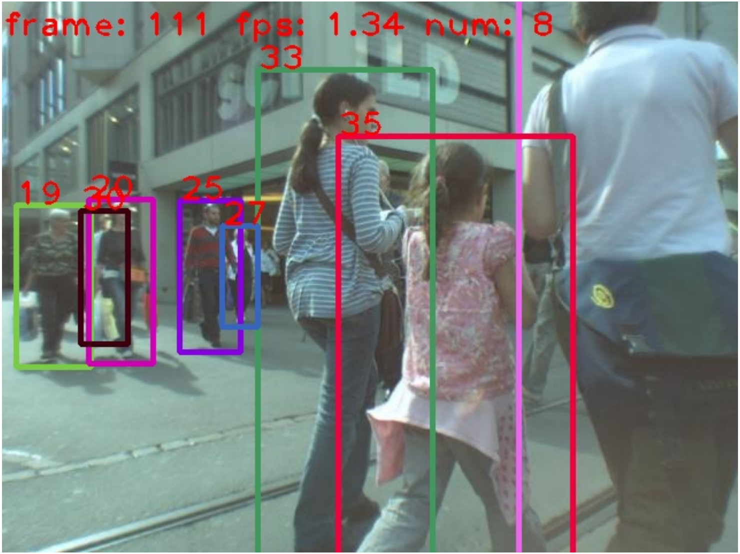
MOT Example: Approximation in Value Space



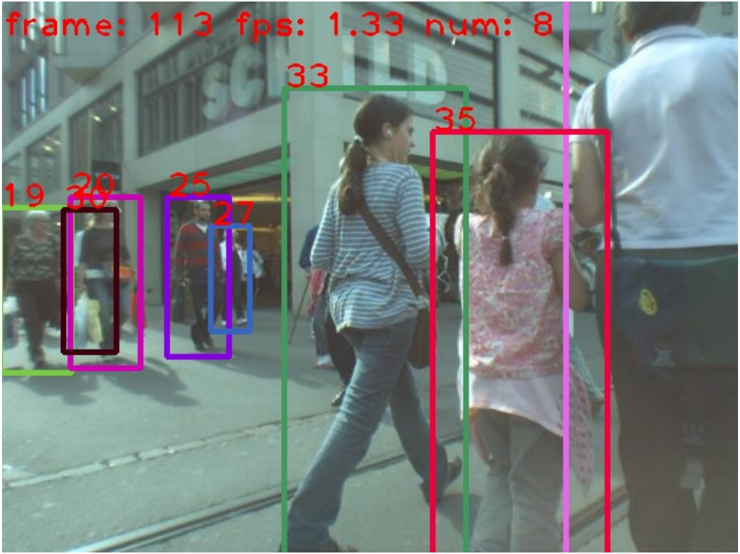

frame: 107 fps: 1.36 num: 7
33
35
34
19
20
36


frame: 109 fps: 1.35 num: 7
33
35
19
20
25
36

MOT Example: Approximation in Value Space

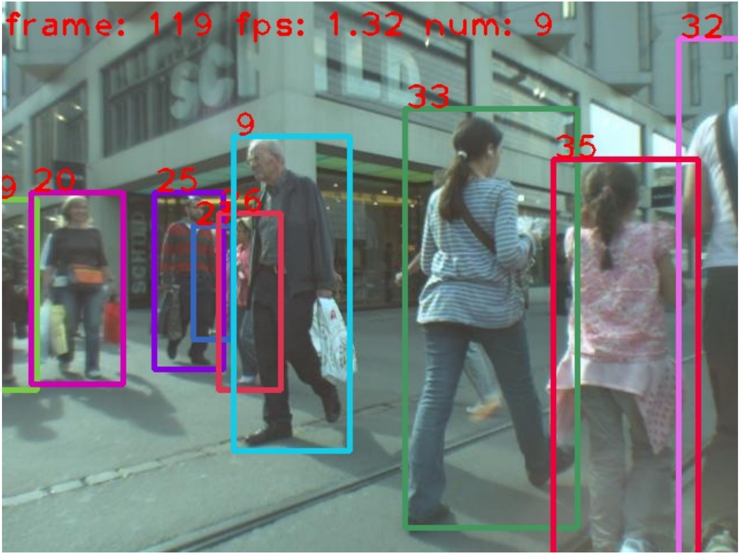


MOT Example: Approximation in Value Space

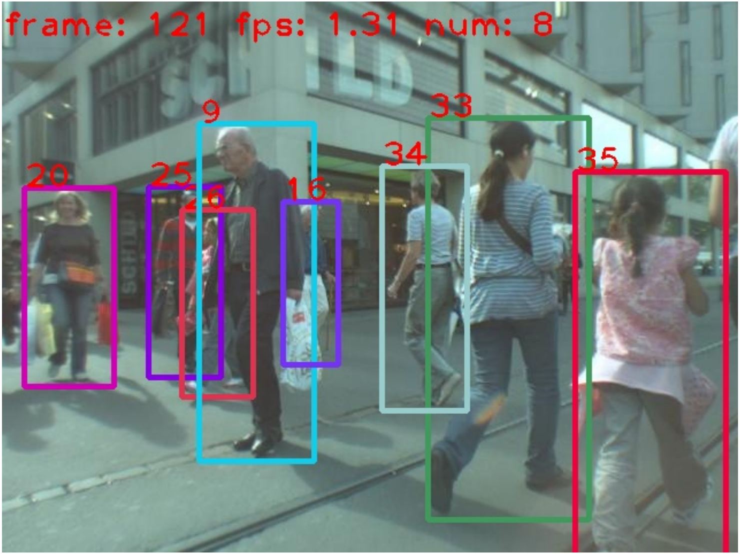



frame: 115 fps: 1.32 num: 10
33
9
34
35
9
20
25
27
26

MOT Example: Approximation in Value Space


frame: 119 fps: 1.32 num: 9
32
33
9
35
20
25

MOT Example: Approximation in Value Space




frame: 123 fps: 1.30 num: 9
9
20
26
16
34
28
33
35

MOT Example: Approximation in Value Space

Outline

1. Model Predictive Control as Approximation in Value Space
2. Computing Most Likely Sequence of a Language Model
3. Addressing Multiple Object Tracking/Data Association Problem
4. Approximation in Value Space with Fine-Tuned Language Model (if time permits)

The Potential of Language Model in Approximation in Value Space

LLM Name	Developer	Release Date	Access	Parameters
GPT-4o	OpenAI	May 13, 2024	API	Unknown
Claude 3	Anthropic	March 14, 2024	API	Unknown
Grok-1	xAI	November 4, 2023	Open-Source	314 billion
Mistral 7B	Mistral AI	September 27, 2023	Open-Source	7.3 billion
PaLM 2	Google	May 10, 2023	Open-Source	340 billion
Falcon 180B	Technology Innovation Institute	September 6, 2023	Open-Source	180 billion
Stable LM 2	Stability AI	January 19, 2024	Open-Source	1.6 billion, 12 billion
Gemini 1.5	Google DeepMind	February 2nd, 2024	API	Unknown

Figure: From `https://explodingtopics.com/blog/list-of-llms`

- There are growing list of language models with impressive capabilities
- Can a given language model act as a base policy or function $\tilde{J}$ used in approximation in value space for a generic task?
- Can fine-tuning (further training with small amount of data for a short time) improve the performance of the policy obtained from approximation in value space?
- We will use chess as our test-bed

The Potential of Language Model in Approximation in Value Space

LLM Name	Developer	Release Date	Access	Parameters
GPT-4o	OpenAI	May 13, 2024	API	Unknown
Claude 3	Anthropic	March 14, 2024	API	Unknown
Grok-1	xAI	November 4, 2023	Open-Source	314 billion
Mistral 7B	Mistral AI	September 27, 2023	Open-Source	7.3 billion
PaLM 2	Google	May 10, 2023	Open-Source	340 billion
Falcon 180B	Technology Innovation Institute	September 6, 2023	Open-Source	180 billion
Stable LM 2	Stability AI	January 19, 2024	Open-Source	1.6 billion, 12 billion
Gemini 1.5	Google DeepMind	February 2nd, 2024	API	Unknown

Figure: From `https://explodingtopics.com/blog/list-of-llms`

- There are growing list of language models with impressive capabilities
- Can a given language model act as a base policy or function $\tilde{J}$ used in approximation in value space for a generic task?
- Can fine-tuning (further training with small amount of data for a short time) improve the performance of the policy obtained from approximation in value space?
- We will use chess as our test-bed

The Potential of Language Model in Approximation in Value Space

LLM Name	Developer	Release Date	Access	Parameters
GPT-4o	OpenAI	May 13, 2024	API	Unknown
Claude 3	Anthropic	March 14, 2024	API	Unknown
Grok-1	xAI	November 4, 2023	Open-Source	314 billion
Mistral 7B	Mistral AI	September 27, 2023	Open-Source	7.3 billion
PaLM 2	Google	May 10, 2023	Open-Source	340 billion
Falcon 180B	Technology Innovation Institute	September 6, 2023	Open-Source	180 billion
Stable LM 2	Stability AI	January 19, 2024	Open-Source	1.6 billion, 12 billion
Gemini 1.5	Google DeepMind	February 2nd, 2024	API	Unknown

Figure: From `https://explodingtopics.com/blog/list-of-llms`

- There are growing list of language models with impressive capabilities
- Can a given language model act as a base policy or function $\tilde{J}$ used in approximation in value space for a generic task?
- Can fine-tuning (further training with small amount of data for a short time) improve the performance of the policy obtained from approximation in value space?
- We will use chess as our test-bed

The Potential of Language Model in Approximation in Value Space

LLM Name	Developer	Release Date	Access	Parameters
GPT-4o	OpenAI	May 13, 2024	API	Unknown
Claude 3	Anthropic	March 14, 2024	API	Unknown
Grok-1	xAI	November 4, 2023	Open-Source	314 billion
Mistral 7B	Mistral AI	September 27, 2023	Open-Source	7.3 billion
PaLM 2	Google	May 10, 2023	Open-Source	340 billion
Falcon 180B	Technology Innovation Institute	September 6, 2023	Open-Source	180 billion
Stable LM 2	Stability AI	January 19, 2024	Open-Source	1.6 billion, 12 billion
Gemini 1.5	Google DeepMind	February 2nd, 2024	API	Unknown

Figure: From `https://explodingtopics.com/blog/list-of-llms`

- There are growing list of language models with impressive capabilities
- Can a given language model act as a base policy or function $\tilde{J}$ used in approximation in value space for a generic task?
- Can fine-tuning (further training with small amount of data for a short time) improve the performance of the policy obtained from approximation in value space?
- We will use chess as our test-bed

Computational Study Based on Pythia and GPT4

- We collected $32,000$ data point from Stockfish (an expert software of chess). Each data is a pair given as

  (chess board configuration,score)

- We used the data to fine-tune an open-source language model, Pythia with 410 million parameters (a much inferior model than GPT4), so that it can act as $\tilde{J}$
- In addition, we used GPT4 as alternative choice of $\tilde{J}$
- As comparison, we also applied GPT4 and the fine-tuned Pythia to play chess directly.

Computational Study Based on Pythia and GPT4

- We collected $32,000$ data point from Stockfish (an expert software of chess). Each data is a pair given as

(chess board configuration,score)

- We used the data to fine-tune an open-source language model, Pythia with 410 million parameters (a much inferior model than GPT4), so that it can act as $\tilde{J}$
- In addition, we used GPT4 as alternative choice of $\tilde{J}$
- As comparison, we also applied GPT4 and the fine-tuned Pythia to play chess directly.

Computational Study Based on Pythia and GPT4

- We collected $32,000$ data point from Stockfish (an expert software of chess). Each data is a pair given as

(chess board configuration,score)

- We used the data to fine-tune an open-source language model, Pythia with 410 million parameters (a much inferior model than GPT4), so that it can act as $\tilde{J}$
- In addition, we used GPT4 as alternative choice of $\tilde{J}$
- As comparison, we also applied GPT4 and the fine-tuned Pythia to play chess directly.

- We collected $32,000$ data point from Stockfish (an expert software of chess). Each data is a pair given as

  (chess board configuration,score)

- We used the data to fine-tune an open-source language model, Pythia with 410 million parameters (a much inferior model than GPT4), so that it can act as $\tilde{J}$
- In addition, we used GPT4 as alternative choice of $\tilde{J}$
- As comparison, we also applied GPT4 and the fine-tuned Pythia to play chess directly.

Computational Results

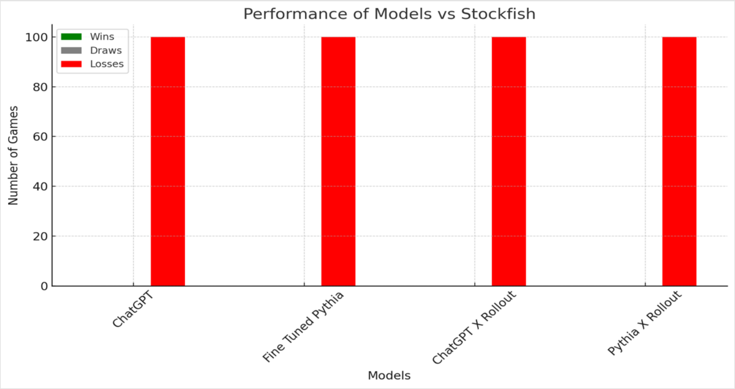


Figure: Collaboration with A. Gundawar

Computational Results

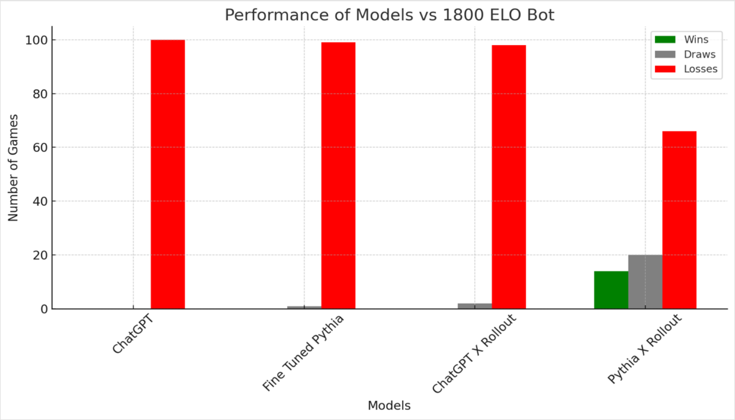


Figure: Collaboration with A. Gundawar

Computational Results

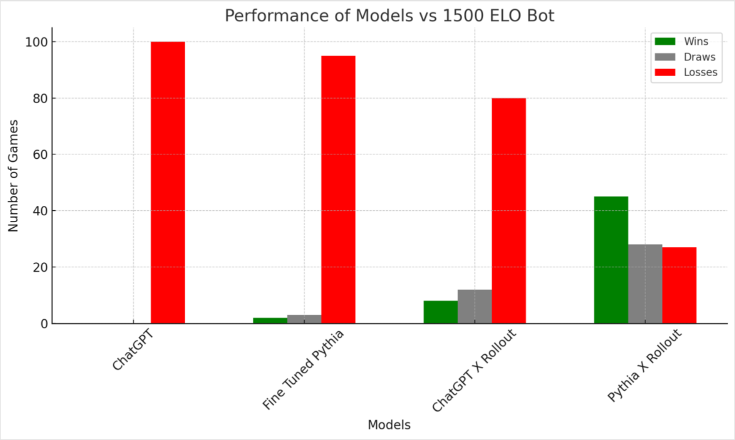


Figure: Collaboration with A. Gundawar