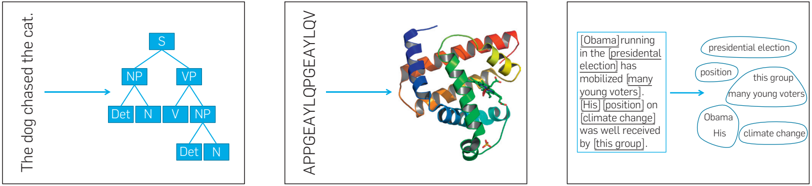
Structured Output Prediction

Andrea Passerini
andrea.passerini@unitn.it

Advanced Topics in Machine Learning and Optimization

Structured Output Prediction: the task



The task

- The input is (typically) a structured object
- The output is also a structured-object (rather than a scalar) e.g.:
  - A sequence (part-of-speech tagging, protein secondary structure prediction)
  - A tree (parse-tree prediction)
  - A graph (link detection, protein 3D structure prediction)

Image from Joachims et al, 2009

Structured Output Prediction: the issue

The issue

- Standard supervised learning learns a function

$$f : \mathcal{X} \to \mathcal{Y}$$

- However the space of candidate outputs is huge (exponential in the number of output variables, or even infinite)
- The problem cannot be formalized as multiclass classification

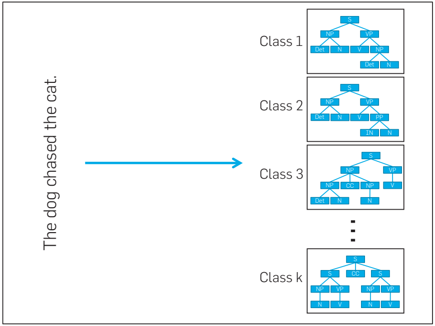
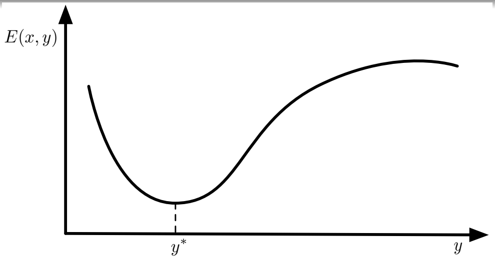


Image from Joachims et al, 2009

Structured Output Prediction: approaches

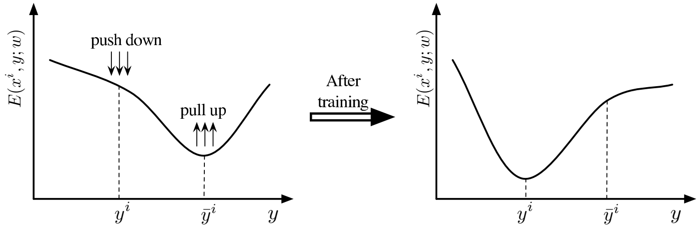


Energy-based models

$$y^* = \mathrm{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- An energy function predicts the energy of each input-output pair
- Prediction is achieved by getting minimal energy output for a given input
- Inference methods are needed to solve the argmin problem (*learning with inference*)

Energy-based models



Learning

- Adjust weights of energy function to drive correct output to have minimal energy
- Based on loss functions between correct output and incorrect ones
- Typically focus on *most offending incorrect answer*:

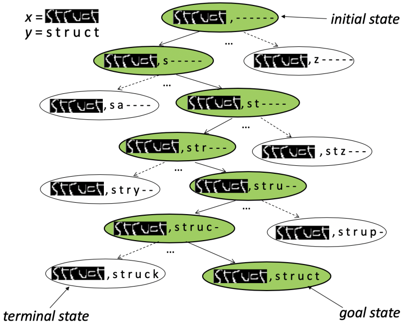
$$\bar{y}^i = \mathrm{argmin}_{y \in \mathcal{Y}, y \neq y^i} E(x^i, y^i; w)$$

Image adapted from LeCun et al. 2006

Structured Output Prediction: approaches

Search-based models

- State-space search process
- Initial state with empty output
- Heuristic function to choose next state (partial output)
- Terminal states are states with complete output
- No need for global inference algorithm (*learning for inference*)

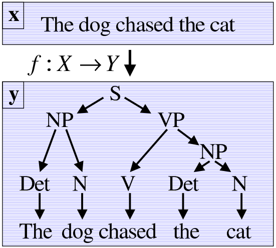


Search-based models

learning

- Adjust weights of heuristic function to have high score for correct moves given current state
- *on-trajectory* training, current state is always a correct one.
- *off-trajectory* training, current state is highest scoring state even if incorrect

Energy-based models: Structured SVM



$$\Psi(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ \vdots \\ 0 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{array}{c} S \to NP\,VP \\ S \to NP \\ NP \to Det\ N \\ VP \to V\ NP \\ \\ Det \to dog \\ Det \to the \\ N \to dog \\ V \to chased \\ N \to cat \end{array}$$

Joint input-output feature map

$$f(x,y) = \mathbf{w}^T \Psi(x,y) = -E(x,y)$$

- Joint input-output feature map $\Psi(x,y)$
- Features capture interaction between input and output variables and between output variables among themselves
- Energy function is a linear function of the feature map
- The function can be kernelized

Image from Tsochantaridis et al., 2005

Structured SVM: learning

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

subject to:

$$\mathbf{w}^T\Psi(x_i, y_i) - \mathbf{w}^T\Psi(x_i, y') \geq \Delta(y_i, y') - \xi_i$$
$$\forall i, y' \neq y_i$$

Max-margin formulation

- $\Delta(y_i, y')$ is the cost for predicting y' instead of y_i (structured-output loss)
- The formulation aims at separating correct predictions from incorrect predictions with a large margin
- Hard to solve directly (exponential number of constraints!!)

Structured SVM: learning

Cutting plane algorithm

1. Initialize weights and constraints $S_i = \emptyset \; \forall i$
2. While constraint added
    1. For each example i

$$\begin{aligned} \xi_i &= \max_{y' \in S_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i) \\ \xi_i^{new} &= \max_{y' \neq y_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i) \end{aligned}$$

    2. If $\xi_i^{new} - \xi > \epsilon$
    3. Add constraint and update S_i
    4. retrain

Alternatives

- Stochastic subgradient descent
- Block-coordinate Frank-Wolfe optimization

Structured SVM: inference

(Loss augmented) argmax inference

- inference at prediction time

$$y^* = \text{argmax}_{y \in \mathcal{Y}} \mathbf{w}^T \Psi(x, y)$$

- loss augmented inference at training time (most offending incorrect answer)

$$\bar{y}' = \text{argmax}_{y' \neq y_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i)$$

Approaches

- Viterbi algorithm for sequence labelling
- CYK algorithm for parse tree prediction
- Loopy belief propagation (approximate)
- Amortized inference (use previous solutions to speed up related inference tasks)

Structured SVM: PROs and CONs

PROs

- Max-margin approach
- Guarantees on number of iterations (depends on ϵ, independent on number of output structures)
- Can deal with arbitrary constrains on output structure

CONs

- Inefficient, (loss augmented) inference required at every training iteration
- The function to be learned is complex, high-order feature typically required (making inference even more expensive)

Search-based models: ordered vs unordered

Ordered search space

- Fixed ordering of decisions (e.g., left-to-right decisions in sequences)
- Classifier-based structured prediction (reduction to multi-class classification task)

Unordered search space

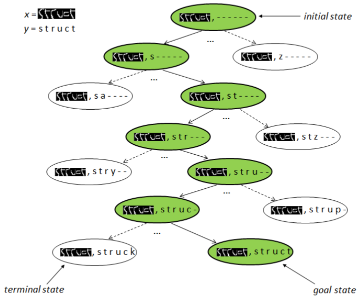
- Learner dynamically orders decisions
- Easy-first approach (make easy decisions first)

Search-based models: classifier-based

Setting

- Ordered search space
- Reduction to multi-class classification on next decision
- Training examples:
  - input is set of outputs up to position t
  - output is correct output for position $t + 1$
- *imitation learning* (training examples as expert demonstrations)

Classifier-based structured prediction: exact imitation



Input | Output

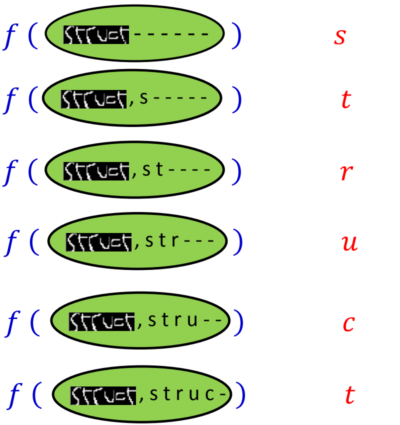
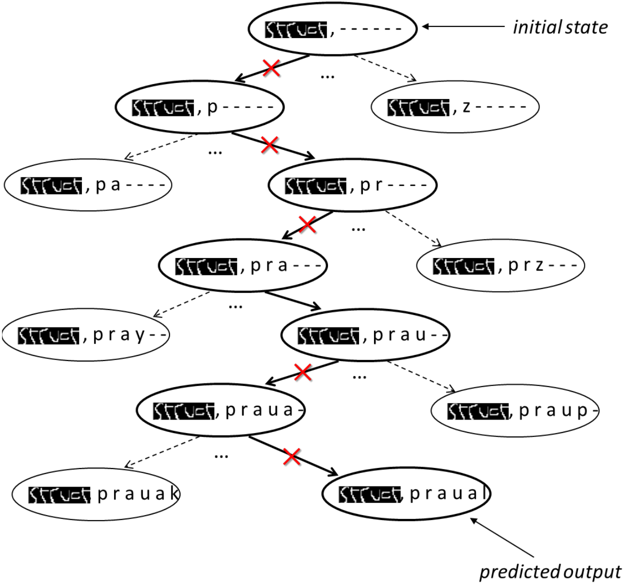


Image from Fern et al., 2016

Exact imitation problem: error propagation



Error propagation

Problem

- Errors in early decisions propagate to down-stream ones
- System is not trained to deal with decisions given incorrect states

Solution

- Generate trajectories using current policy
- Use optimal policy to generate optimal next states given states visited by current policy

DAgger (Dataset Aggregation)

The algorithm

1. Collect training set $\mathcal{D}$ of N trajectories using ground-truth policy π^*
2. Repeat
    1. $\pi \leftarrow \textsc{LearnClassifier}(\mathcal{D})$
    2. Collect set of states $\mathcal{S}$ along trajectories computed using π
    3. For each $s \in \mathcal{S}$
        1. $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, \pi^*(s))\}$
3. Return π

Search-based models: easy-first approach

CONs of classifier-based approaches

- Need to define an ordering over output variables
- Some decision are harder than others $\rightarrow$ fixed ordering can be suboptimal

Easy-first approach: rationale

- Make easy decisions first to constraint harder ones
- Learn to dynamically order decisions
- Analogous to constraint satisfaction algorithms

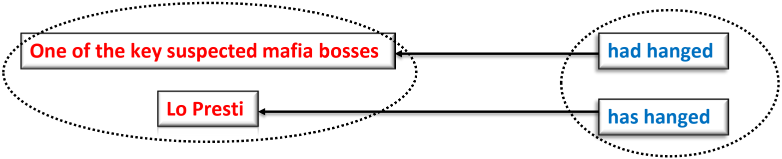
Example: Cross-document coreference

One of the key suspected mafia bosses arrested yesterday **had hanged** himself.

Doc 1

Police said **Lo Presti** **has hanged** himself.

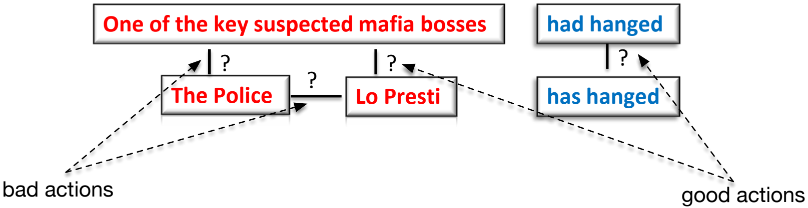
Doc 2



Hard

Easy

Easy-first approach: inference



Easy action first

- State s is partial solution
- Set of possible actions $a \in A(s)$ from a state (no ordering)
- Action scoring function $f(s, a) = \mathbf{w}^T \Psi(s, a)$
- Proceed making highest scoring (most-confident) action first

Easy-first approach: learning

Easy-first policy learning

while not termination condition **do**
    for $(x, y) \in \mathcal{D}$ **do**
        $s \leftarrow I(x)$
        while not $\textsc{isTerminal}(s)$ **do**
            $a_p \leftarrow \max_{a \in A(s)} w^T \Psi(s, a)$
            if $a_p \in B(s)$ **then**
                $\textsc{update}(w, G(s), B(s))$
            end if
            $a_c \leftarrow \textsc{chooseAction}(A(s))$
            $s \leftarrow$ Apply a_c on s
        end while
    end for
end while

Easy-first policy learning

$$\text{UPDATE}(w, G(s), B(s))$$

Variants

- Highest scoring good action better than highest scoring bad action (perceptron update)
- Highest scoring good action better than all bad actions

$$a_c \leftarrow \text{CHOOSEACTION}(A(s))$$

Variants

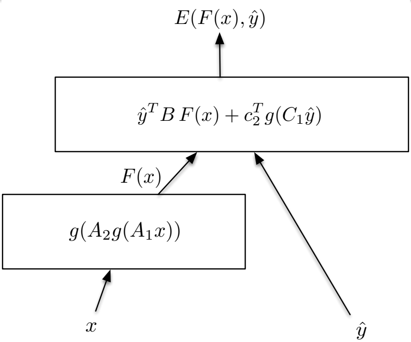
- Choose highest scoring *good* action ($a_c \in G(s)$, on-trajectory training)
- Choose highest scoring action ($a_c \in G(s) \cup B(s)$, off-trajectory training)

Combining energy-based and search-based approaches

HC-search framework

- Generate high-quality candidate complete outputs with search-based approach (H = search heuristic)
- Score candidates with energy function and select minimal energy output (C = cost/energy function)

Deep energy-based methods



Structured Prediction Energy Networks (SPEN)

- Energy function modelled as a deep network
- Replaces outputs $y \in \{0, 1\}^L$ with relaxations $\hat{y} \in [0, 1]^L$
- Training by gradient descent over weights using structured loss (e.g. as in structured SVM)
- Inference by gradient descent over $\hat{y}$ (+ rounding if needed)

SPEN

PROs

- Efficient inference by gradient descent
- No need to pre-specify input-output features (input-output representation learning)

CONs

- No algorithmic guarantees (local optimization of energy)
- No management of explicit constraints
- No support for hard constraints

Deep search-based methods

Transformers for content generation

- **Autoregressive models**: predict next token given input tokens + currently generated ones
- **Attention-based models**: use attention to learn token embeddings that depend on other tokens in the context
- Trained with combinations of:
  - self-supervised learning
  - supervised fine tuning
  - reinforcement learning with human feedback

Memory augmented Transformer

Transformer problems

- Cannot access up-to-date information
- Storing all knowledge in the model parameters does not scale
- Enriching prompts with potential knowledge (RAG) also does not scale

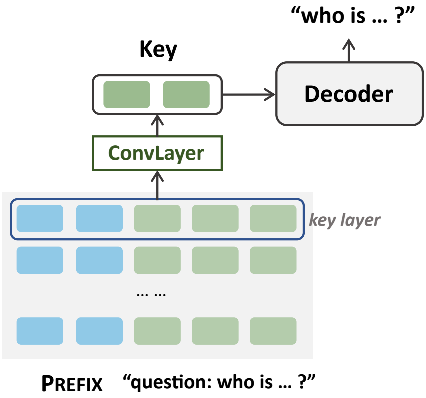
Solution

- Give transformers ability to use a *key-value memory*
- Encode Q&A pairs in the memory

Memory augmented Transformer: key embedding

Procedure

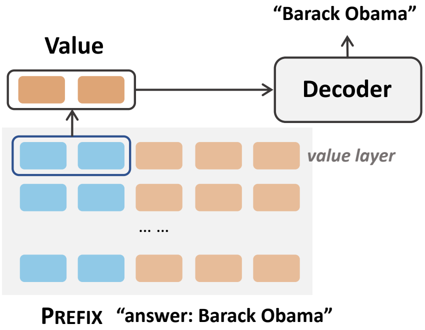
1. concatenate PREFIX with query
2. pass through encoder, get k^{th} layer
3. pass through conv layer, get prefix as key



Memory augmented Transformer: value embedding

Procedure

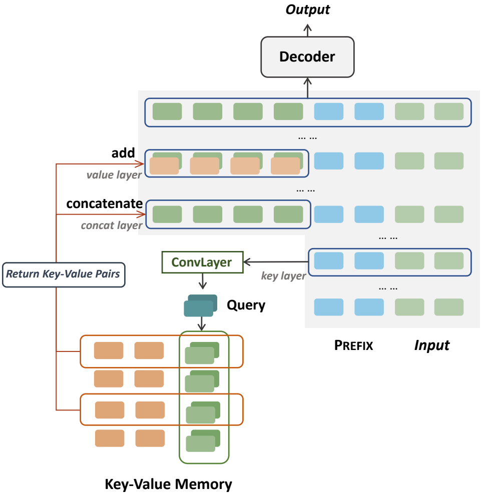
1. concatenate PREFIX with answer
2. pass through encoder, get v^{th} layer
3. get prefix as value



Memory augmented Transformer: memory retrieval

Procedure

1. encode query same as key embedding
2. perform inner product with memory keys
3. retrieve top-k key-value pairs
4. keys are sorted by similarity and prepended at layer c
5. values are sorted by similarity and added at layer v



Toolformer: self-learning to use tools

Transformer problems

- Problems in performing precise calculations
- Tendency to hallucinate facts

Solution

- Give transformers ability to use *external tools*
- Allow them to learn when and how to use tools (with little human annotation)

Toolformer: examples

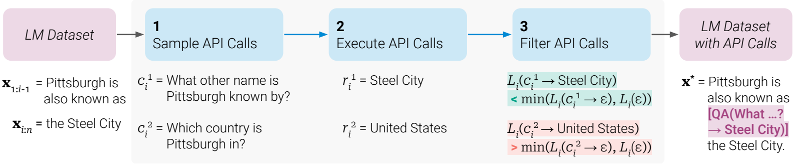
The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

Toolformer: overview



Few-shot driven dataset expansion

1. Sample API calls
2. Execute API calls
3. Filter API calls
4. Finetune model

Toolformer: sample API calls - 1

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

Input: x

Output:

PROMPT($\mathbf{x}$)

Create API-specific prompt

$$PROMPT(\mathbf{x})$$

Toolformer: sample API calls - 2

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

Input: Pittsburgh is also known as the Steel City

Output: Pittsburgh is

[PROMPT('Pittsburgh is also known as the Steel City'), 'Pittsburgh is']

Sample candidate API-call positions according to

$$p_i = P('[' \mid PROMPT(\mathbf{x}), x_{1:i-1})$$

Toolformer: sample API calls - 3

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

Input: Pittsburgh is also known as the Steel City

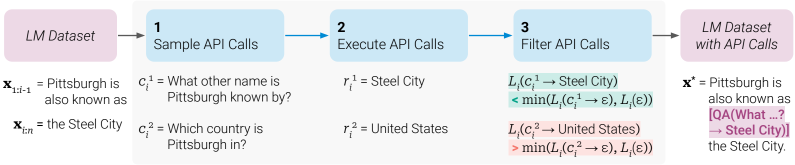
Output: Pittsburgh is also known as [

[PROMPT('Pittsburgh is also known as the Steel City'), 'Pittsburgh is', '[']

Sample candidate API calls for *i* from the sequence

$$[PROMPT(\mathbf{x}), x_{1:i-1}, '['] \quad up\ to\ ']'$$

Toolformer: execute, filter, finetune



Execute, filter, finetune

1. Execute API for each sampled call
2. Filter results based on whether they reduce loss for subsequent tokens
3. Finetune model with expanded dataset including retained calls (+ results)

Toolformer: inference

API-augmented inference

1. Plain decoding until '$\rightarrow$'
2. Call API
3. Insert response + ']'
4. Continue decoding

GeLaTo: **Ge**nerating **La**nguage with **T**ractable **Co**nstraints

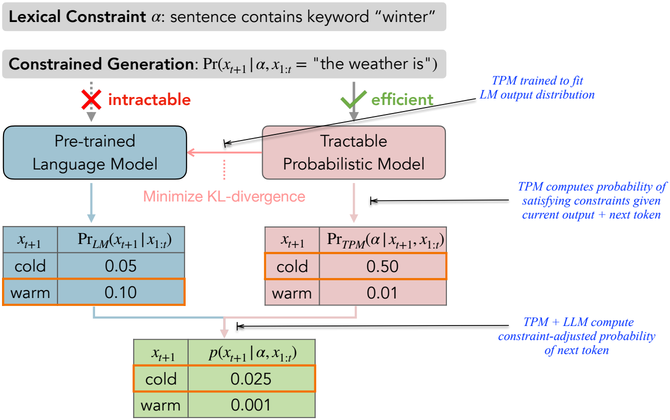
Transformer problems

- Autoregressive models cannot enforce (non-local) constraints
- Search-based solutions are very expensive

Solution

- Combine tranformer with a tractable probabilistic model (TPM)
- Efficiently enforce constraints on the TPM

GeLaTo: architecture



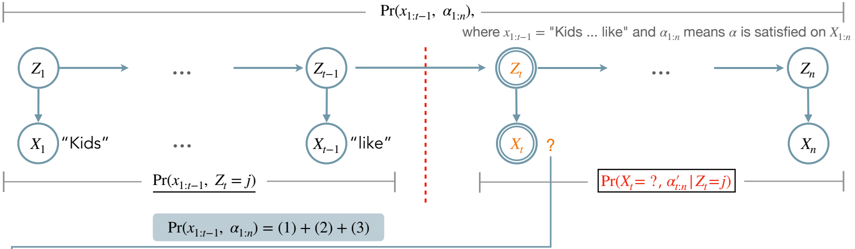
x_{t+1}	$\Pr_{LM}(x_{t+1} \mid x_{1:t})$
cold	0.05
warm	0.10

x_{t+1}	$\Pr_{TPM}(\alpha \mid x_{t+1}, x_{1:t})$
cold	0.50
warm	0.01

x_{t+1}	$p(x_{t+1} \mid \alpha, x_{1:t})$
cold	0.025
warm	0.001

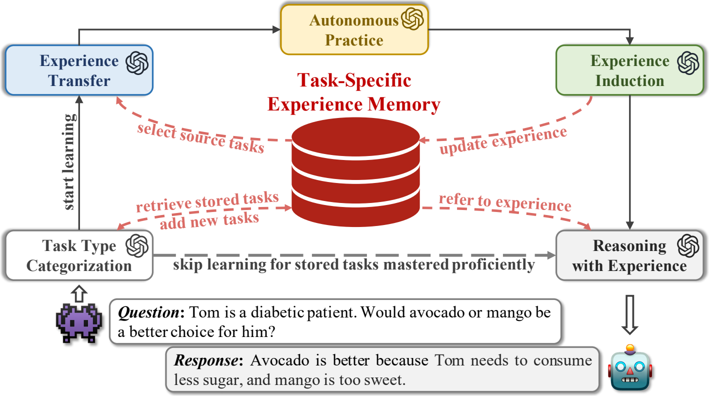
GeLaTo: example of inference

Lexical Constraint $\alpha = \big(I(\text{like eating}) \lor I(\text{soccer})\big) \land I(\text{like working})$

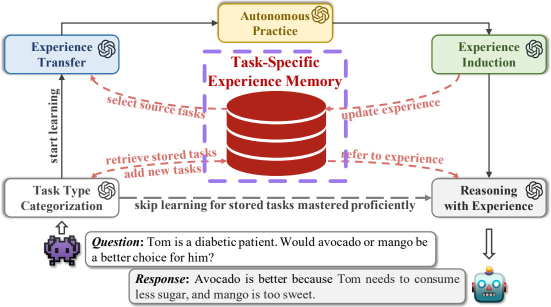


X_t	Recurrence	α'
"eating"	(1) $\Pr(x_{1:t-1}, X_t = \text{"eating"}, \alpha'_{t:n}) = \sum_j \underline{\Pr(x_{1:t-1}, Z_t = j)} \cdot \boxed{\Pr(X_t = \text{"eating"}, \alpha'_{t:n} \mid Z_t = j)}$	$I(\text{like working})$
"working"	(2) $\Pr(x_{1:t-1}, X_t = \text{"working"}, \alpha'_{t:n}) = \sum_j \underline{\Pr(x_{1:t-1}, Z_t = j)} \cdot \boxed{\Pr(X_t = \text{"working"}, \alpha'_{t:n} \mid Z_t = j)}$	$I(\text{like eating}) \lor I(\text{soccer})$
else	(3) $\Pr(x_{1:t-1}, X_t \neq \text{"eating"}, X_t \neq \text{"working"}, \alpha'_{t:n})$ $= \Pr(x_{1:t-1}, \alpha'_{t:n}) - \Pr(x_{1:t-1}, X_t = \text{"eating"}, \alpha'_{t:n}) - \Pr(x_{1:t-1}, X_t = \text{"working"}, \alpha'_{t:n})$ $= \sum_j \underline{\Pr(x_{1:t-1}, Z_t = j)} \cdot \left(\boxed{\Pr(\alpha'_{t:n} \mid Z_t = j)} - \boxed{\Pr(X_t = \text{"eating"}, \alpha'_{t:n} \mid Z_t = j)} - \boxed{\Pr(X_t = \text{"working"}, \alpha'_{t:n} \mid Z_t = j)}\right)$	$\big(I(\text{like eating}) \lor I(\text{soccer})\big)$ $\land\ I(\text{like working})$

Self-evolving GPT: self-learning from experience



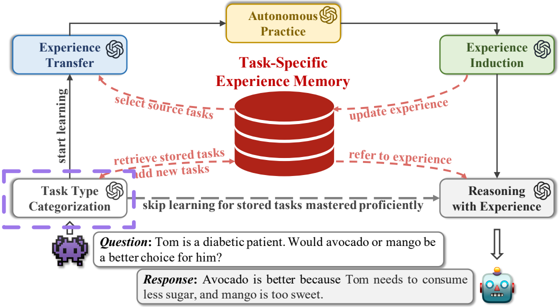
Self-evolving GPT: self-learning from experience



Experience Memory

- Starts empty
- Stores tasks after addressing them
- Stores task name, description and experience
  - Procedure: steps for handling task
  - Suggestions: how to better accomplish task / avoid errors

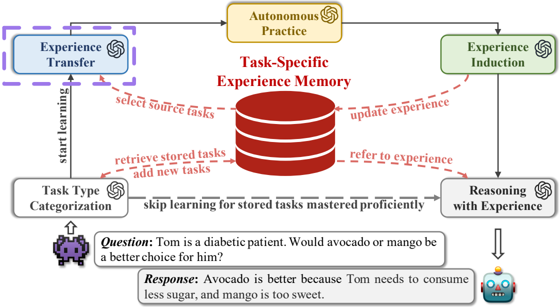
Self-evolving GPT: self-learning from experience



Task Type Categorization

1. retrieve similar tasks from memory
2. if match found
    1. retrieve task from memory
    2. if task adequately learned **skip learning**
    3. otherwise **start learning**
3. otherwise, add new task to memory

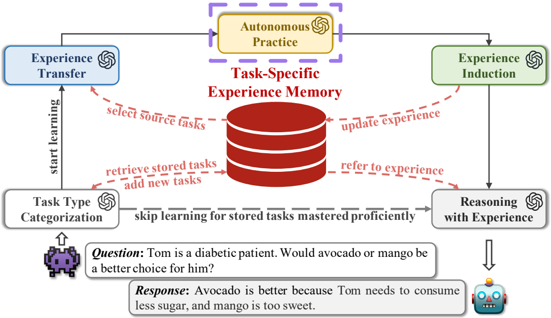
Self-evolving GPT: self-learning from experience



Experience Transfer

1. step-by-step experience transfer (prompt-based)
   1. understand differences
   2. identify shared experience
   3. rephrase it for target task
2. merge transferred experience with task experience

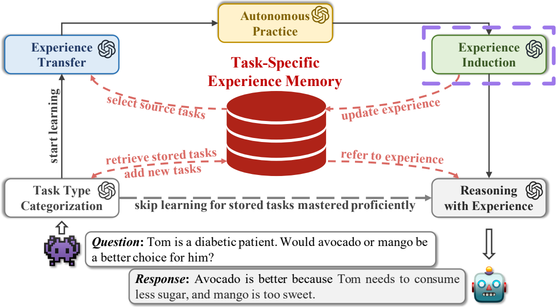
Self-evolving GPT: self-learning from experience



Autonomous Practice

1. retrieve web documents related to question
2. generate task-specific question related to document
3. verify correctness from document

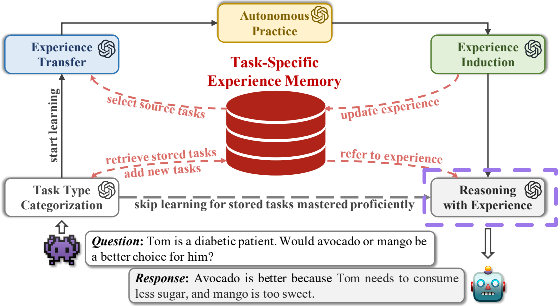
Self-evolving GPT: self-learning from experience



Experience Induction

1. summarize new experience for current task
   1. summarize commonalities between correct examples
   2. identify patterns in incorrect examples
   3. generate task-solving insights
2. merge induced experience with existing experience

Self-evolving GPT: self-learning from experience



Reasoning with Experience

1. Retrieve experience for current task
2. Address task based on experience

References

Bibliography

- Deshwal, A.; Doppa, J. R.; and Roth, D., *Learning and inference for structured prediction: A unifying perspective*, in IJCAI 2019.
- LeCun, Y.; Chopra, S.; Hadsell, R.; Huang, F. J.; and et al., *A tutorial on energy-based learning*, in Predicting Structured Data, MIT Press.
- Joachims, T.; Hofmann, T.; Yue, Y.; and Yu, C.-N., *Predicting structured objects with support vector machines*, in Comm. of the ACM, 2009.
- Daumé, H.; Langford, J.; and Marcu, D., *Search-based structured prediction*, in Machine Learning, 2009.
- Ross, S.; Gordon, G.; and Bagnell, D., *A reduction of imitation learning and structured prediction to no-regret online learning*, in AISTATS, 2011.
- Belanger D. and McCallum, A., *Structured prediction energy networks*, in ICML 2016.
- Wu Y., Zhao Y., Hu B., Minervini P., Stenetorp P., and Riedel S., *An Efficient Memory-Augmented Transformer for Knowledge-Intensive NLP Tasks*, EMNLP 2022.
- Schick T., Dwivedi-Yu J., Dessì R., Raileanu R., Lomeli M., Zettlemoyer L., Cancedda N., Scialom T., *Toolformer: Language Models Can Teach Themselves to Use Tools*, NeurIPS, 2023.
- Zhang H., Dang M., Peng N., and Van Den Broeck G., *Tractable control for autoregressive language generation*, ICML 2023.
- Gao, J. ; Ding, X. ; Cui, Y. ; Zhao, J. ; Wang, H. ; Liu, T., *Self-Evolving GPT: A Lifelong Autonomous Experiential Learner*. ACL, 2024.