SENAI Lab

Deep Path

RL for Knowledge Graph Reasoning

Chaitanya- ED21B018

Aim

- Learning reasoning of large scale knowledge graph
- RL for multi-hop reasoning
- TranslationE Embeddings
- REINFORCE policy graident update

Related Work

Multi-hop learning

Involves navigating through several interconnected entities to arrive at a conclusion.
2 approaches:

1. Embedding based reasoning: By representing entities and relationships in a continuous vector space, the algorithm can perform operations that infer new knowledge based on the proximity of these embeddings.
2. Link-based: focuses on the direct traversal of the graph's edges to find paths that connect the entities involved in a query. It relies on the structural properties of the graph to derive answers.

Implementation- Reasoning process modelled as a Markov Decision Process.
Actor Critic Network:

$$\Delta\theta \propto \nabla \log \pi(a|s) \cdot (R + \gamma V(s') - V(s))$$

In the paper, a controllable multi-hop reasoning approach is introduced

Path Ranking Algorithm

Core Principles:

- Path-based Reasoning: PRA operates by treating paths between entity pairs as features. Each path is evaluated based on its relevance and contribution to the relationship between the entities involved. The algorithm ranks these paths to determine which are most indicative of the desired relationship.
- Soft Inference Procedure: Unlike hard inference methods that strictly adhere to predefined rules, PRA employs a soft inference approach. This allows for more flexibility in reasoning, accommodating uncertainty and ambiguity in the paths.

Math Formulation:

$$\text{score}(e; s) = \sum_{P \in P_\ell} h_{s,P}(e)\theta_P$$

- If P is an empty path:

$$h_{s,P}(e) = \begin{cases} 1 & \text{if } e = s \\ 0 & \text{otherwise} \end{cases}$$

- If P is non-empty, it is defined as:

$$h_{s,P}(e) = \sum_{e' \in \text{range}(P')} h_{s,P'}(e') \cdot P(e|e'; R_\ell)$$

where $P(e|e'; R_\ell)$ is the probability of transitioning from entity e' to entity e along the relation R_ℓ.

$$O(\theta) = \sum o_i(\theta) - \lambda_1|\theta|_1 - \lambda_2|\theta|_2$$

o_i (theta) is per instance objective function based on the predicted relevance of entity pairs.
h_sp(e) is path feature function that represents probability of reachiing entity e from s following path P
theta is weight associated with path, learned during training.

Reinforcement Learning

To capture semantic info-
Transaltion based embedding

States: $\mathbf{s}_t = (\mathbf{e}_t, \mathbf{e}_{target} - \mathbf{e}_t)$

$$r_{\text{GLOBAL}} = \begin{cases} +1, & \text{if the path reaches } e_{targ} \\ -1, & \text{otherwise} \end{cases}$$

$$r_{\text{EFFICIENCY}} = \frac{1}{length(p)}$$

$$r_{\text{DIVERSITY}} = -\frac{1}{|F|} \sum_{i=1}^{|F|} cos(\mathbf{p}, \mathbf{p}_i)$$

Proof

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla\pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_\pi\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right] && \text{(replacing } a \text{ by the sample } A_t \sim \pi\text{)} \\
&= \mathbb{E}_\pi\left[G_t\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right], && \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)\text{)}
\end{aligned}
$$

Why it works

This update has an intuitive appeal. Each increment is proportional to the product of a return Gt and a vector, the gradient of the probability of taking the action actually taken divided by the probability of taking that action. The vector is the direction in parameter space that most increases the probability of repeating the action At on future visits to state St.
The former makes sense because it causes the parameter to move most in the directions that favor actions that yield the highest return. The latter makes sense because otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return.

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

MONTE-CARLO REINFORCE

Algorithm

As a stochastic gradient method, REINFORCE has good theoretical convergence properties. By construction, the expected update over an episode is in the same direction as the performance gradient.
It may be of high variance and thus produce slow learning.

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}.$$

Training Pipeline

1

Supervised policy Learning

Since we have a large relation set, we train with randomised breadth-first search.
For each (e_src, e_enty), 2 side BFS to find correct paths .

2

Exploring right

Vanilla BFS prefers shorter paths, leaving the meaningful ones. Hence added random intermediate node and 2 BFS bn src and entty.Concatenated paths used for training

3

Retraining with rewards

Stochastic poiicy followed to prevent getting stuck.
Reinforced with reward update after supervised learning

Restore parameters θ from supervised policy;
for *episode* $\leftarrow 1$ **to** N **do**
  Initialize state vector $s_t \leftarrow s_0$
  Initialize episode length $steps \leftarrow 0$
  while $num\_steps < max\_length$ **do**
    Randomly sample action $a \sim \pi(a|s_t)$
    Observe reward $\mathcal{R}_t$, next state s_{t+1}
    `// if the step fails`
    if $\mathcal{R}_t = -1$ **then**
      Save $< s_t, a >$ to $\mathcal{M}_{neg}$
    if *success or* $steps = max\_length$ **then**
      break
    Increment num_steps
  `// penalize failed steps`
  Update θ using
  $g \propto \nabla_\theta \sum_{\mathcal{M}_{neg}} \log \pi(a = r_t|s_t; \theta)(-1)$
  if *success* **then**
    $R_{total} \leftarrow \lambda_1 r_{\text{GLOBAL}} + \lambda_2 r_{\text{EFFICIENCY}} + \lambda_3 r_{\text{DIVERSITY}}$
    Update θ using
    $g \propto \nabla_\theta \sum_t \log \pi(a = r_t|s_t; \theta) R_{total}$

Verification

BiDirectional Path Search

Initialise at the source and target entity and follow both the paths in the opposite direction, till find an intersection in both sets

Link Prediction

Link Prediction: Of target entitites
The performance is measured using Mean Average Precision (MAP), which evaluates the quality of the ranking of correct target entities higher in the predicted list.
Using two datasets: FB15K-237 and NELL-995.

Fact prediction

This task involves determining whether a given triple (entity, relation, entity) is valid.
For this, both true and generated false triples are ranked, and the model's ability to rank true triples higher indicates better performance.

Networks

Policy, Value and Q-Network of each 3 layers of NN.

- Policy Network: Drives the agent's behavior by suggesting which actions to take based on the current state.
- Value Network: Provides a baseline for the policy gradient methods, helping to reduce variance in training.
- Q-Network: Estimates the quality of actions, allowing the agent to select the action with the highest expected reward.

Supervised

LTraining Loop: Iterates through the training samples:

- For each episode, it creates an environment and retrieves a sample.
- Attempts to find valid paths between entities using a teacher function (BFS-based).
- Updates the policy network with state-action pairs extracted from successful episodes.

Scoring

- Average Precision (AP) Calculation: For each model (TransE, TransR, RL), computes the average precision by ranking the test triples and calculating the proportion of correctly ranked positive triples. This involves checking each ranked item and computing the proportion of correct predictions up to each position.

Main

teacher: This function uses BFS to find paths between entities e1 and e2 using intermediate entities. It cleans paths by removing loops (duplicate entities), and converts paths into sequences of state-action pairs suitable for training the RL agent.

path_clean: Removes cycles in paths by identifying and eliminating duplicate entities, leaving only the shortest valid paths.

Implementation

Results

Link Prediction

- DeepPath achieved the highest MAP scores in both datasets, outperforming PRA and embedding methods. This demonstrated its superior ability to leverage multi-hop paths for accurate reasoning.
- Path Analysis: DeepPath discovered more concise, informative paths compared to PRA, which often yielded redundant paths.

Fact Prediction

Similar to link prediction, DeepPath outperformed the baseline methods, showing that its reinforcement learning framework with reward-guided path selection is robust for determining the validity of facts within the KG.

Path length

DeepPath effectively filtered out less relevant paths, focusing on the most predictive ones, which was facilitated by its continuous state representation and reward system.

The supervised pre-training is crucial as it helps the agent to avoid poor convergence which is common in high-dimensional action spaces of knowledge graphs.

Thank You

Chaitanya