CS 410 - Technology Review

Ashish Kumar, Aruna Neervannan, Abhisek Jana

Overview

We are working on implementing **DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning** using **TensorForce**, an open source reinforcement learning library built on top of **TensorFlow**. This paper talks about a novel reinforcement learning framework to learn multi-hop relational paths using a policy based agent with states based on knowledge graph embeddings. As per part of the Technology Review we have explained our understanding of the paradigm of Reinforcement Learning and how it is being used for Knowledge Graph reasoning using state-of-the-art reinforcement learning libraries (TensorForce).

What is Reinforcement Learning

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals. This approach is much more focused on goal-directed learning from interaction and is called **Reinforcement Learning**.

Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. Essentially, these are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics are the three most important distinguishing features of the reinforcement learning problem

- Being closed-loop in an essential way
- Not having direct instructions as to what actions to take
- Where the consequences of actions, including reward signals, play out over extended time periods

Reinforcement learning is also different from unsupervised learning (machine learning), which is typically about finding structure hidden in collections of unlabeled data. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure (maximizing reward is the primary function of the agent). Uncovering structure in an agent's experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning agent's problem of maximizing a reward signal. We therefore consider reinforcement learning to be a third machine learning paradigm, alongside supervised learning and unsupervised learning.

In **reinforcement learning (RL)** there's no answer key, but your reinforcement learning **agent** still has to decide how to act to perform its task. In the absence of existing training data, the agent learns from experience. It collects the training examples (“this action was good, that action was bad”) through **trial-and-error** as it attempts its task, with the goal of maximizing long-term **reward**.

Examples of Reinforcement Learning

A good way to understand reinforcement learning is to consider some of the examples and possible applications that have guided its development.

- Make a humanoid robot walk
- Play many different Atari games better than humans
- A master chess player makes a move. The choice is informed both by planning – anticipating possible replies and counter replies and by immediate, intuitive judgements of the desirability of particular positions and moves.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.

These examples share features that are so basic that they are easy to overlook. All involve interaction between an active decision-making agent and its environment, within which the agent seeks to achieve a goal despite uncertainty about its environment. The agent’s actions are permitted to affect the future state of the environment (e.g., the next chess position, the robot’s next location and the future charge level of its battery), thereby affecting the options and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning.

Elements of Reinforcement Learning

Beyond the agent and the environment, one can identify four main sub elements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment.

A **policy** defines the learning agent’s way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states.

A **reward** signal defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent’s sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent.

A **value function** specifies what is good in the long run, in contrast to a reward signal which indicates what is good in the immediate sense. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

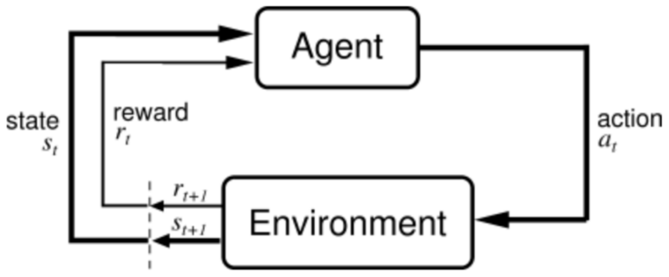
The final element of some reinforcement learning systems is a **model** of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward.

This is a sequential decision-making process

- **Goal**: select actions to maximize total future reward

- **Actions** may have long term consequences
- **Reward** may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward

Agent and the Environment



At each step t the agent:

- Executes action a_t
- Receives observation o_t
- Receives scalar reward r_t

The environment:

- Receives action a_t
- Emits observation o_{t+1}
- Emits scalar reward r_{t+1}

t increments at env. step

The mathematical framework for defining a solution in reinforcement learning scenario is called **Markov Decision Process**. This can be designed as:

- Set of states, s
- Set of actions, a
- Reward function, r(s, a) which tells us which states and actions are better
- Policy, π
- Value, v

We have to take an action (a) to transition from our start state to our end state (s). In return getting rewards (r) for each action we take. Our actions can lead to a positive reward or negative reward.
The set of actions we took define our policy π and the rewards we get in return defines our value v. We maximize the rewards by choosing the correct policy.

In other words, we maximize $E(r_t \mid \pi, s_t)$ for all possible values of S for a time t.

Exploration and Exploitation

One of the interesting problems that arises when using Reinforcement Learning is the tradeoff between exploration and exploitation. If an agent has tried a certain action in the past and got a decent reward, then repeating this action is going to reproduce the reward. In doing so, the agent is exploiting what it knows to receive a reward. On the other hand, trying other possibilities may produce a better reward, so exploring is definitely a good tactic sometimes. Without a balance of both **exploration** and **exploitation** the RL agent will not learn successfully. The most common way to achieve a nice balance is to try a variety of actions while progressively favoring those that stand out as producing the most reward.

Types of RL Algorithms

Policy gradients

These methods rely on optimizing the parameterized policy function with respect to the return by gradient descent.

Value-Iteration based schemas

These methods estimate the value function or Q-function of the optimal policy (with no explicit policy specified).

Actor-critic learning

These methods combine the benefits of policy gradient and value-iteration models. The network will estimate both a value function or Q-function and a policy. The agent uses the value estimate (the critic) to update the policy (the actor) more intelligently than traditional policy gradient methods.

Model-based RL

The agent uses a model of the environment to learn a value function and improve its performance.

TensorFlow

TensorFlow is a powerful open source software library, developed by Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research. TensorFlow combines the computational algebra of compilation optimization techniques, making easy the calculation of many mathematical expressions where the problem is the time required to perform the computation.

Overview:

- The core library is suited to a broad family of machine learning techniques, not “just” deep learning.
- Linear algebra and other internals are prominently exposed.
- In addition to the core machine learning functionality, TensorFlow also includes its own logging system, its own interactive log visualizer, and even its own heavily engineered serving architecture.
- The execution model for TensorFlow differs from Python's scikit-learn, or most tools in R.
- TensorBoard is a utility to visualize different aspects of machine learning

TensorFlow has one of the easiest installations of any platform, bringing machine learning capabilities squarely into the realm of casual tinkerers and novice programmers. Meanwhile, high-performance features, such as—multi-GPU support, make the platform exciting for experienced data scientists and industrial use as well. TensorFlow also provides a reimagined process and multiple user-friendly utilities, such as TensorBoard, to manage machine learning efforts. Finally, the platform has significant backing and community support from the world's largest machine learning powerhouse--Google. All this is before even considering the compelling underlying technical advantages, which we'll dive into later.

Currently, TensorFlow is supported on Linux, Mac, and Windows. Most of the original core code for TensorFlow is written in C++. While TensorFlow can run on the CPU, most algorithms run faster if processed on the GPU, and it is supported on graphics cards with Nvidia Compute Capability v4.0+ (v5.1 recommended). Popular GPUs for TensorFlow are Nvidia Tesla architectures and Pascal architectures with at least 4 GB of video RAM. To run on a GPU, you will also need to download and install the Nvidia Cuda Toolkit and also v 5.x + (https://developer.nvidia.com/cuda-downloads). Some of the most popular python libraries (Scipy, Numpy, and Scikit-Learn) works seamlessly with TensorFlow.

Main Features:

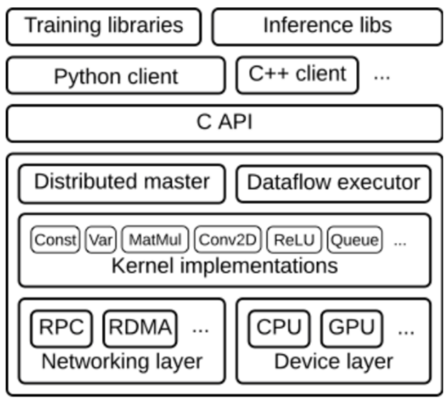
- Defining, optimizing, and efficiently calculating mathematical expressions involving multi-dimensional arrays (tensors).
- Programming support of deep neural networks and machine learning techniques.
- Transparent use of GPU computing, automating management and optimization of the same memory and the data used. You can write the same code and run it either on CPUs or GPUs. More

specifically, TensorFlow will figure out which parts of the computation should be moved to the GPU.

- High scalability of computation across machines and huge data sets.

High Level Architecture:

- **Client**:
  - Defines the computation as a dataflow graph.
  - Initiates graph execution using a session
- **Distributed Master**
  - Prunes a specific subgraph from the graph, as defined by the arguments to Session.run().
  - Partitions the subgraph into multiple pieces that run in different processes and devices.
  - Distributes the graph pieces to worker services.
  - Initiates graph piece execution by worker services.
- **Worker Services** (one for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, etc.).
  - Send and receive operation results to and from other worker services.
- **Kernel Implementations**
  - Perform the computation for individual graph operations.



TensorFlow is popular:

- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- Visualization (TensorBoard is da bomb)
- Checkpoints (for managing experiments)
- Auto-differentiation autodiff (no more taking derivatives by hand)
- Large community (> 10,000 commits and > 3000 TF-related repos in 1 year)
- Awesome projects already using TensorFlow

Natural Language Processing/Text Analysis using TensorFlow

- TensorFlow is also very powerful for Text Analysis tasks (Word Embedding, Skip-gram Model etc.)
- There are also specific tools for Natural Language Processing, such as **SyntaxNet**
  - SyntaxNet is a framework for what's known in academic circles as a syntactic parser, which is a key first component in many NLU systems. Given a sentence as input, it tags each word with a part-of-speech (POS) tag that describes the word's syntactic function, and it determines the syntactic relationships between words in the sentence, represented in the dependency parse tree. These syntactic relationships are directly related to the underlying meaning of the sentence in question.
    Please refer the below link for more details.
    https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

- TensorFlow’s robust RNN (Recurrent Neural Network) API could be used advance text processing and machine intelligence.

TensorForce

TensorForce is a python-based reinforcement learning (RL) library built on top of TensorFlow. As noted in previous section, TensorFlow is an open source machine learning software library developed by Google.

TensorForce library was created by three researchers: Michael Schaarschmidt, Alexander Kuhnle and Kai Fricke. Their goal is to provide portable computation graphs for RL regardless of context whether it is robotics, autonomous vehicles or finance. Reinforcement learning is evolving rapidly but it is also very hard to learn. They main objective of this library is to modularize various elements of machine learning. Reinforcement learning is an active area of research. However, the practical usage of RL based applications has demonstrated following issues consistently.

- Most implementations are custom built for a specific application. It means that RL code and simulation environment code are typically tightly coupled making it harder to separate when it is time to exit prototyping phase and take it to production.
- We have observed most implementations have custom coded the neural network for training. While it is convenient to do that for a researcher, it becomes very difficult for a future researcher to plugin a different type of neural network or simply remove it.
- The **state and action options** for an AI environment are typically unique which typically leads to implementation of an agent good enough for that particular environment. The authors of this library identified this problem early on and decided to build an agent capable of handing any number of states and/or actions.
- **TensorForce is based on TensorFlow**. TensorFlow implementation is typically specific to a hardware platform – for example, there is a generic CPU only implementation, GPU implementation, ActivePython's version and Intel Nervana's version optimized Intel Xeon and Intel Xeon Phi processors. This custom implementation forces researchers to do significant amount of custom implementation for their hardware. The TensorForce library provides an abstract layer and allows user to declaratively define their hardware and takes care of necessary under-the-hood custom libraries.

TensorForce allows researchers to use declarative framework to implement deep reinforcement learning algorithms. Instead of describing agent is described using configuration object. Similarly, the neural network for training is defined as an ordered list of layers. It is easier to why the object based declarative interface facilitates a modular approach for RL algorithms. At the time of writing this document, following RL algorithms are available out of the box.

- Random Baseline agent
- Vanilla policy gradient
- Trust Region Policy
- Deep Q-Learning/ double deep Q-Learning
- Normalized advantage functions
- Asynchronous advantage actor-critic (A3C) – there is no A3CAgent. A3C describes a mechanism for asynchronous updates.

It is important to note that TensorForce enforces the separation between model and agent. The Agent class provides APIs to use RL while the model class implements the core RL algorithms. Another key feature of this library is the ability to define various components as a JSON object. For example, the neural network configuration can be stored as a JSON object in a separate file. This separation enforces and encourages modular framework and provides an easy way to swam out neural networks without having to rewrite the code or logic.

The choice of TensorForce library for the DeepPath project is a logical choice. Researchers have implemented the environment, actions, rewards, agent and states in custom python libraries. TensorForce API library will allow us to modularize the implementation of DeepPath. Our work will empower future researchers to swap out select components without having to worry about doing significant code rewrite. In the next section, we will provide a brief overview of OpenAI gym – an open source platform for building and testing agents in a gaming environment. Detailed API documentation for TensorForce is available here. TensorForce continues to be in active development with rapid commit cycles on github.

OpenAI Gym

OpenAI Gym is an open source platform for creating, evaluating and benchmarking artificial agents in a game environment. OpenAI Gym focuses on the episodic setting of reinforcement learning, where the agent's experience is broken down into a series of episodes. In each episode, the agent's initial state is randomly sampled from a distribution, and the interaction proceeds until the environment reaches a terminal state. The goal in episodic reinforcement learning is to maximize the expectation of total reward per episode, and to achieve a high level of performance in as few episodes as possible. OpenAI Gym contains a collection of Environments (POMDPs), which are community developed and supported and is expected to grow over time. Detailed documentation on the setup and usage can be obtained here.

OpenAI Gym consists of two parts:

1. The gym open-source library: a collection of test problems — environments — that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms.
2. The OpenAI Gym Service: a site and API allowing people to meaningfully compare performance of their trained agents.

Knowledge Graph

Knowledge graph, a concept introduced by Google, mainly describes real world entities and their interrelations, organized in a graph covering various topical domains. Knowledge is about collecting information about objects in the real world. The object could be a person, book, movie or any other type of objects. Knowledge graphs help improve search results by understanding what the user is searching for. It acquires and integrates information into an ontology (which describes the semantics of the data) and applies a reasoner to derive new knowledge.

Knowledge graphs being actual graphs, in the proper mathematical sense, allow for the application of various graph-computing techniques and algorithms (for example, shortest path computations, or network analysis), which add additional intelligence over the stored data. The ontology can be extended and revised as new data arrives. With these semantics defined, the results could go deeper and broader resulting in unexpected findings.

Summary

Our final project for CS 410 is implementing **DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning** using TensorForce. The RL system will consist of two parts:

- The external environment E which specifies the dynamics of the interaction between the agent and the knowledge graph This environment is modeled as Markov Decision Process (MDP).
- The RL agent is represented as a policy network $\pi_\theta(s, a) = p(a \mid s; \theta)$ which maps the state vector to a stochastic policy thus using a policy based RL methodology. The neural network consists of two hidden layers, each followed by a rectifier nonlinearity layer (ReLU). The output layer is normalized using a softmax function available in TensorForce.

Team Overview

Aruna is the author of the following sections:

- What is Reinforcement Learning, Knowledge Graph, OpenAI gym, Summary.

Abhisek is the author of the following section:

- TensorFlow

Ashish is the author of the following section:

- TensorForce

Overview and Team Overview sections are coauthored by all three of us.