Sandia National Laboratories

SAND2020-7558D

Automated Incorporation of Machine Learning (AIM)

Olawale E. Salaudeen
University of Illinois at Urbana-Champaign, PhD Computer Science
Sandia National Laboratories, Albuquerque, NM; U.S. Department of Energy
Manager: Michael J. Haass (Org. 09365), Mentor: Eric Goodman (Org. 09365)

Abstract

The efforts of researchers and growth of compute in the last decade has yielded many Machine Learning (ML) algorithms that have been proven to outperform traditional statistical algorithms and sometimes human experts on a variety of tasks. Accordingly, the applications of ML in industries such as communication, healthcare, and defense grow rapidly. One consequence of this growth is the development of standard architectures that are proven to be most effective for certain types of tasks, standard libraries for implementing said architectures, and a seemingly endless and diverse supply of examples via opensource repositories, such as Github. We utilize the corpus of opensource ML code to learn an ML model that can automatically generate ML code from inputs containing information about the type of task we would like to solve.

Background

Sequence to sequence models are designed to solve a class of problems where the input is a sequence and the output is also a sequence. The class of models have seen much success in tasks such as text summarization, audio to text, etc. The most prevalent models used for these class of problems are recurrent neural networks (RNN), however, these methods suffer from mismatches in training and real world evaluation metrics, and exposure bias. Though techniques have been developed for mitigating these often crippling limitations of RNNs, a class of models, Reinforcement Learning (RL), are designed to overcome these limitations by default. The recent success and developments in RL make them a prime candidate for automatic machine learning code generation.

Results and Discussion

The goal of this current iteration of the project is to accurately complete a line in machine learning code; a simple example would be – *input:* {*import, tensorflow, as*}, *output:* {*tf, \n*}. One of the challenges of this project has been reducing the complexity of the problem, specifically the action space. Since the dataset used to train the model is a combination of python code written by different users for different tasks, there is a lot of noise in the data. We try to increase the signal-to-noise ratio by limiting our action space, or vocabulary, to tokens that occur more frequently than a certain threshold. In addition, there are issues of class-imbalance and many contradictory samples. For instance we could have *input:* {*from tensorflow import*} and *output: (1)* {*keras, \n*} *(2)* {*estimator, \n*}, both of which are valid sequences. These all have negative implications on the optimization landscape of the problem and decrease the learnability of the task. Currently, our model seems to get stuck at a bad local minima where it is effectively a constant predictor of the most frequent token in the data.

Method

Our training data consists of 1 million Tensorflow code pulled from Github. The data is tokenized by spaces and non-alphanumeric character – comments are removed. The training data is composed of input sequence and output sequence; each sample is consecutive tokens of length *l*. The input sequence is the first *m* tokens of the sample sequence and the output sequence is the last *o* tokens, where the sample sequence length *l* = *m* + *o*.

We utilize the Deep Q Network (DQN) architecture, also known as Actor-Critic with Q Function Estimation, for automatic machine learning code generation. DQN is a special case of the more general Actor-Critic model with advantage function:

$$A_\pi(s_t, y_t) = Q_\pi(s_t, y_t) - V_\pi\,(s_t)$$

where $\mathrm{Q}_\pi = r_t + \gamma E_{s_{t'} \sim \pi(s_{t'}|s_t)}[V_\pi(s_{t'})]$, the expected discounted reward of the state-action pairs, (s_i, y_i) respectively. In the DQN however, $V_\pi\,(s_t) = 0$.

The actor is a sequence to sequence model implemented as an encoder and attention decoder utilizing long short-term memory layers (LSTM) – trained with crossentropy loss + the future rewards estimated by the critic.

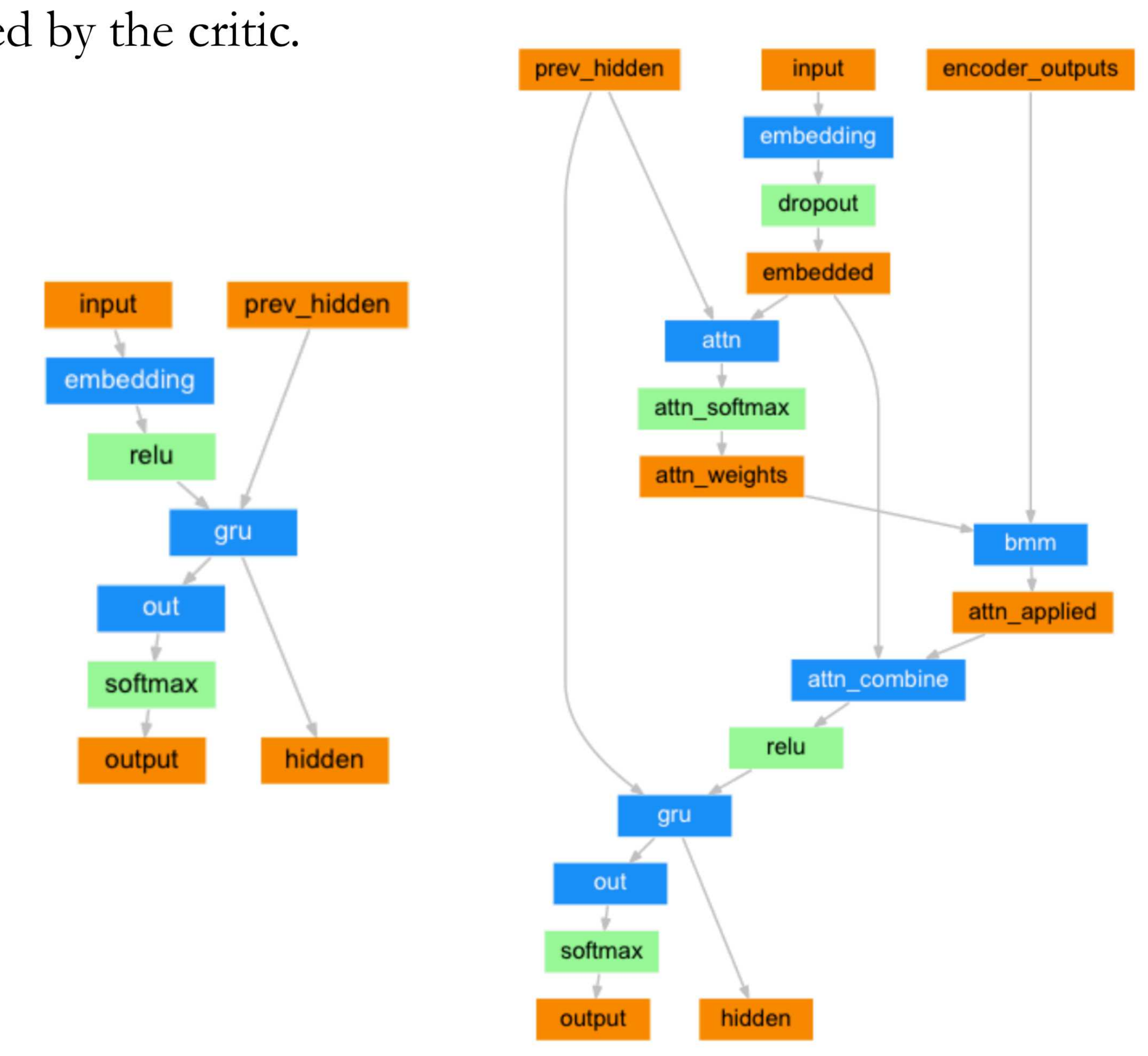

Figure 1: *Left: Encoder, Right: Attention Decoder. GRU layers are replaced with LSTM Layers. Figures source:* https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

We learn a critic as a regression model where the hidden units of the decoder are states and the output sequence are actions, (s_i, y_i) respectively:

$$L(\psi) = \frac{1}{2}\sum_i ||Q_\psi(s_i, y_i) - q_i||^2$$
$$q_i = r_t + \gamma \max_{y'} Q_\psi(s'_i, y'_i)\,.$$

References

Keneshloo, Yaser & Shi, Tian & Ramakrishnan, Naren & Reddy, Chandan. (2019). Deep Reinforcement Learning for Sequence-to-Sequence Models. IEEE Transactions on Neural Networks and Learning Systems. PP. 1-21. 10.1109/TNNLS.2019.2929141.