(June 12th, 2018)

Project Title (Tentative): Developing a medical Natural Language Generation system using Deep Learning and Reinforcement Learning

Abstract

We aim to build a prototype framework of Natural Language Generation (NLG) for the medical field leveraging recent advances in Deep Learning and Reinforcement Learning. By defining the task of NLG as sequence generation process, we implement two baselines, one built upon Long Short-Term Memory (LSTM)-based Recurrent Neural Network (RNN) and the other with Monte Carlo Tree Search (MCTS). After implementing the two baselines using PyTorch, we plan to develop other variants based on MCTS for improving the performance of the baselines, while focusing roles of evaluation metric, target corpus, and algorithmic benefits, etc.

Project Overview

* Developing effective generative models for human language generation
* Important component potentially used for the tasks such as image captioning, dialogue system, Q & A, etc
* Focusing on “how to say”, given “what to say”

Overarching objectives

* Understanding and implementing mechanisms on how RNN-based models generate sequences
* Developing a variant of RNN-based model with MCTS-based decoding
* Implementing and seeking improvements of MCTS-based NLG
* Applications using clinical notes and the comparison of developed approaches

I. References

* LSTM RNN
  + Intro for the modern RNN-based models: Karpathy’s blog <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
  + Main Reference for LSTM-based sequence generation (baseline model I) : Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.
  + Wen, T. H., Gasic, M., Mrksic, N., Su, P. H., Vandyke, D., & Young, S. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv* *preprint arXiv*:1508.01745
* MCTS
* MCTS-based Natural Language Generation (baseline model II) : Kumagai, K., Kobayashi, I., Mochihashi, D., Asoh, H., Nakamura, T., & Nagai, T. (2016). Human-like Natural Language Generation Using Monte Carlo Tree Search. In *Proceedings of the INLG 2016 Workshop on Computational Creativity in Natural Language Generation* (pp. 11-18).
* Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1-43.
* Deep Reinforcement learning
* Some sort of Introduction (Only for fun and your knowledge)
* <https://deepmind.com/>
* (Just one hit randomly) <https://www.wired.com/story/greedy-brittle-opaque-and-shallow-the-downsides-to-deep-learning/>
* (More to come)

II. Research Project

1. Introduction of Deep Learning

* Artificial Neural Network models (since 2006, emphasized in its deep architecture)
  + End-to-end training with backpropagation
  + Scalable for large datasets
* Great performance for human-like perception problems (vision, NLP, audio, speech, etc). \*biological sequences such as genomic and proteins. \*some interests on graph datasets
* Machine Learning framework with representation learning, offering effective methods for unsupervised learning and other advanced methods such as meta learning, transfer learning, active learning, multi-task learning, multi modal learning, etc
* Deep Reinforcement Learning
  + Deep Learning topics with Reinforcement Learning techniques
  + Reinforcement Learning combined with Deep Learning-based function approximation
* Types of Deep Learning
  + Basic models such as DNN, RNN, and CNN, and any hybrid or complex architectures extending such basic models
  + RNN – Long Short-Term Memory (LSTM)
    - Seq2Seq model
    - Encoder-decoder model
    - (Soft) Attention model

1. Introduction of Monte Carlo Tree Search and Reinforcement Learning
   * MDP vs. POMDP

* Markov Decision Process as a cornerstone framework for Reinforcement Learning
* POMDP (Partially Observable MDP)
* Monte-Carlo Tree Search
* Proven to be effective for complicated planning problems such as the game of Go
* Exploration and exploitation dilemma
* Viable solution for POMDP and large state and action space problems
* Not fully appreciated for its capability and possible variations for many challenging problems

1. Overall comparative summary on the two (DL and RL) approaches for text sequence generation

(underway)

|  |  |  |
| --- | --- | --- |
|  | LSTM-based models | MCTS-based models |
| Known approaches for generating sequences | Seq2Seq  Encoder-decoder  (soft) Attention |  |
| Key aspects | * Generate a token sequentially (Autoregressive model) * Potential discrepancy between training sets and a test sequence |  |
| Semantic condition |  |  |

1. Natural Language Generation for medical information

* Two baseline models
* LSTM-based approach (DL): RNN as a sequence generating neural network model
* MCTS-based approach (RL): MCTS as a sequence planning model
* Improving MCTS-based approach with DRL
* Medical text corpus
* MIMIC database
* Model design
* Assuming that “what to say” is given, “how to say” is focused

1. LSTM-based Sequence Generation

* Examples
  1. <https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html> (Pytorch)
  2. <https://github.com/karpathy/char-rnn> (torch)

(Case I) Classification

<https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html>

(checking points)

* classification with softmax regression and Negative Log Likelihood (NLL)
* Basic pattern for a deep learning program
* Data pre-processing
* Model construction
* Train
* Evaluation
* Test
* Data set
* Pre-processing for preparing the input tensor, e.g. [# of tokens in seq., # of data in each minibatch, size of input (# of features)]
* N-fold cross validation, training data and test data
* Training
* SGD, ADAM, RMSprop, and many
* Drop out and batch normalization
* Performance metric (accuracy)
* Precision and recall
* Sensitivity and specificity
* F1
* ROC and AUC

(Case II) Sequence Generation

a. Simple character generation

<https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html>

b. Seq2Seq, encoder-decoder, and attention

<https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html>

<https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb>

(misc.)

<https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb>

<https://github.com/IBM/pytorch-seq2seq>

<https://github.com/MaximumEntrophy/Seq2Seq-PyTorch>

<https://github.com/eladhoffer/seq2seq.pytorch>

LSTM-based Baseline Model I

(Main References)

* Wen, T. H., Gasic, M., Mrksic, N., Su, P. H., Vandyke, D., & Young, S. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv* *preprint arXiv*:1508.01745
* Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv*:1308.0850

(Implementations) SC-LSTM

<https://github.com/hit-computer/SC-LSTM>

<https://github.com/shawnwun/RNNLG>

<https://github.com/shawnwun/NNDIAL>

1. Monte Carlo Tree Search-based Natural Language Generation

Baseline Model II

(Main Reference) Kumagai, K., Kobayashi, I., Mochihashi, D., Asoh, H., Nakamura, T., & Nagai, T. (2016). Human-like Natural Language Generation Using Monte Carlo Tree Search. In *Proceedings of the INLG 2016 Workshop on Computational Creativity in Natural Language Generation* (pp. 11-18).

1. Comparison of performance between the two baselines
2. Improving the MCTS-based model

Supp. Technical training

Pre-requisite

* PyTorch
* Spacy
* GitHub
* Jupyter
* Anadonda/PIP (Home Brew for Mac)
* Linux commands: ssh, etc
* Programming Development Environment Tool: ex) Microsoft Visual Studio Code

Note: There are numerous tutorials and YouTube clips you can watch for more information. And, the most efficient way to master and to remember what you know is to repeat the same thing many times!

1. GitHub

<https://www.atlassian.com/git?utm_source=basic-git-commands&utm_medium=link&utm_campaign=git-microsite&_ga=2.75964245.1174987269.1527087707-1877757822.1527087707>

\* Our repository location : <http://github.com/sdaysmerrill/Summer-REU-Project/>

1. Python
   * Shell vs. Jupyter notebook
   * Package management with conda and pip

https://pip.pypa.io/en/stable/

* + Environment with anaconda

<https://conda.io/docs/_downloads/conda-cheatsheet.pdf>

1. Jupyter

1. ssh
   * ssh reu@130.39.94.206

password : your\_first\_name + “this year”

1. PyTorch
   * Tensor : <https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html>
   * Autograd : <https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html>
   * Neural Network : <https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html>
   * Training a classifier : <https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>
   * Other examples : <https://github.com/pytorch/examples> (ex : word\_language\_model, snli)
   * Pytorch github repository : <https://github.com/pytorch>

(Key Ideas)

* Forward calculation with Neural Network model
* Backward (Backpropagation) with automatic differentiation
* Training with Loss minimization (Regression and Classification)
* Stochastic Gradient Decent with batch

RNN model

**import** **torch.nn** **as** **nn**

**class** **RNN**(nn.Module):

**def** \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

super(RNN, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.i2h = nn.Linear(input\_size + hidden\_size, hidden\_size)

self.i2o = nn.Linear(input\_size + hidden\_size, output\_size)

self.softmax = nn.LogSoftmax(dim=1)

**def** forward(self, input, hidden):

combined = torch.cat((input, hidden), 1)

hidden = self.i2h(combined)

output = self.i2o(combined)

output = self.softmax(output)

**return** output, hidden

**def** initHidden(self):

**return** torch.zeros(1, self.hidden\_size)

n\_hidden = 128

rnn = RNN(n\_letters, n\_hidden, n\_categories)

Training

learning\_rate = 0.005 *# If you set this too high, it might explode. If too low, it might not learn*

**def** train(category\_tensor, line\_tensor):

hidden = rnn.initHidden()

rnn.zero\_grad()

**for** i **in** range(line\_tensor.size()[0]):

output, hidden = rnn(line\_tensor[i], hidden)

loss = criterion(output, category\_tensor)

loss.backward()

*# Add parameters' gradients to their values, multiplied by learning rate*

**for** p **in** rnn.parameters():

p.data.add\_(-learning\_rate, p.grad.data)

**return** output, loss.item()

Visualization of Training

a. iterative loss calculation

**import** **time**

**import** **math**

n\_iters = 100000

print\_every = 5000

plot\_every = 1000

*# Keep track of losses for plotting*

current\_loss = 0

all\_losses = []

**def** timeSince(since):

now = time.time()

s = now - since

m = math.floor(s / 60)

s -= m \* 60

**return** '*%d*m *%d*s' % (m, s)

start = time.time()

**for** iter **in** range(1, n\_iters + 1):

category, line, category\_tensor, line\_tensor = randomTrainingExample()

output, loss = train(category\_tensor, line\_tensor)

current\_loss += loss

*# Print iter number, loss, name and guess*

**if** iter % print\_every == 0:

guess, guess\_i = categoryFromOutput(output)

correct = '✓' **if** guess == category **else** '✗ (*%s*)' % category

**print**('*%d* *%d%%* (*%s*) *%.4f* *%s* / *%s* *%s*' % (iter, iter / n\_iters \* 100, timeSince(start), loss, line, guess, correct))

*# Add current loss avg to list of losses*

**if** iter % plot\_every == 0:

all\_losses.append(current\_loss / plot\_every)

current\_loss = 0

b. plotting

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib.ticker** **as** **ticker**

plt.figure()

plt.plot(all\_losses)

F. Spacy

Home Page: <https://spacy.io>

Introduction:

* <https://spacy.io/usage/linguistic-features>
* <https://spacy.io/usage/vectors-similarity>
* <https://spacy.io/usage/training>
* <https://spacy.io/usage/adding-languages> (advanced)

Examples:

* <https://spacy.io/usage/examples>