(June 20th, 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 the baseline model that is an encoder-decoder model with Long Short-Term Memory (LSTM)-based Recurrent Neural Network (RNN) using PyTorch. After the baseline model, we further plan to develop its variant utilizing Monte Carlo Tree Search (MCTS) for the decoding process. We compare this new model with the baseline, understanding potential merits of the new model with respect to the performance in medical NLG, 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 (optional)
* 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

* Base line models
  + LSTM-based approach (DL) : RNN as a sequence generating neural network model
  + MCTS-based approach (RL) : MCTS as a sequence planning model (optional)
* New hybrid model with LSTM-based approach that Incorporates MCTS for decoding
* Medical text corpus
  + MIMIC database
* Model design
  + Assuming that “what to say” is given, “how to say” is focused

1. RNN-based Sequence Generation Examples

E.1. Example for Classification task

<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
  + Optimizers : SGD, ADAM, RMSprop, and many
  + Drop out and batch normalization
* Performance metric (accuracy)
  + Precision and recall
  + Sensitivity and specificity
  + F1
  + ROC and AUC

E.2. Examples for Sequence Generation

1. Simple character generation <https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html> (Pytorch) <https://github.com/karpathy/char-rnn> (torch)
2. Encoder-decoder and attention models <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/MaximumEntropy/Seq2Seq-PyTorch>

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

Suggested script structure

1. Data preprocessing
   1. Loading data files
   2. Training, Validation, and Testing data preparation
   3. Training data into input
2. Building the Model
   1. Encoder
   2. Decoder with Attention module (Bahdanau and Luong)
   3. Testing the model
3. Training
   1. Teacher forcing or Scheduled sampling
   2. Deterministic Training (BLEU)
4. Evaluation and Visualization

1. (In the case of Sequence Generation) Decoding
   1. beam search
   2. random sampling

RNN-based Baseline Model- Encoder-decoder with attention model for semantically conditioned text generation

(Main Reference)

* Wen, T. H., Rojas-barahona, L. M., & Su, P. H. (2015). Toward multi-domain language generation using recurrent neural networks.

(Related works)

* 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*.

|  |
| --- |
| * Wen, T. H., Gasic, M., Mrksic, N., Rojas-Barahona, L. M., Su, P. H., Vandyke, D., & Young, S. (2016). Multi-domain neural network language generation for spoken dialogue systems. *arXiv preprint arXiv:1603.01232*. |

* Tutorial from the Cambridge group : <https://shawnwun.github.io/talks/DL4NLG_20160906.pdf>

(other references)

* Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*
* Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks, NIPS '14
* Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation , 2014
* Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

(Implementations)

Cambridge group repository

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

* Hlstm
* Sclstm
* Encdec
* Knn
* N-gram

SC-LSTM from other and misc

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

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

(Implementation Summary)

* Datasets
  + Restaurant
  + Hotel
  + Laptop
  + Tv
* Model
  + Encoder-decoder architecture with attention model for slot-value pairs
  + Teacher forcing (optionally Scheduled Sampling)
* Training
  + Machine Learning (optionally Deterministic Training)
* Decoding
  + Beam Search (optionally random sampling)
* Evaluation metric
  + BLEU

(Limitations and challenges)

1. Monte Carlo Tree Search-based Natural Language Generation (optional)

(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. New model that incorporates MCTS into RNN-based RNN

* Main considerations

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)

1. 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>