(June 4th, 2018)

Project Title (Tentative) : Toward 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). We also attempt 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.

I. References

* LSTM RNN
  + Karpathy’s blog for RNN : <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*.
  + (More to come)
* 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)

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

(more to come)

* Monte-Carlo Tree Search

(more to come)

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>

(Case II) Sequence Generation

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

LSTM-based Baseline Model I

1. Monte Carlo Tree Search-based Natural Language Generation
2. Comparison of performance between the two baselines
3. 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)