Machine Learning Engineer Nanodegree

Capstone Proposal

Pavel (Pasha) Gyrya

October 29th, 2017

Proposal

Domain Background

In this project I would like to tackle the task of automatic machine translation, inspired by Google paper. The problem of automated text normalization, picked up from Kaggle competition for data scientists seems like a perfect practice grounds to learn and practice the details necessary to successfully execute this task while preparing to tackle many other media-to-sequence translations (e.g. music annotation, speech-to-text transcription, etc.)

Many speech and language applications, including text-to-speech synthesis (TTS) and automatic speech recognition (ASR), require text to be converted from written expressions into appropriate "spoken" forms. This is a process known as text normalization, and helps convert 12:47 to "twelve forty-seven" and $3.16 into "three dollars, sixteen cents."

However, one of the biggest challenges when developing a TTS or ASR system for a new language is to develop and test the grammar for all these rules, a task that requires quite a bit of linguistic sophistication and native speaker intuition.

Problem Statement

In this competition, we were challenged to automate the process of developing text normalization grammars via machine learning, focusing on English. In other words, we are looking at a scalable algorithm that would learn to recommend translations (e.g. step-by-step translations, one word at a time), based on the input sequence of symbols. In this context, the task could be thought of as a classification problem among many classes representing differed words where word recommendation is expected at every step of generating translation text.

I would further narrow this project down to normalizing numeric text (e.g. numbers) so as to setup a scale-able proof-of-concept algorithm and estimate resource needs. Once this scale-able process is setup, it would be straightforward to execute it for other languages and for other text forms offered in the competition.

Datasets and Inputs

As part of the competition, we are provided with a large corpus of text including various form of text, already transcribed into a spoken form and broken down into logical parts that can be transcribed independently. This data is provided as table with before column (containing raw text) and after column (containing normalized text to be predicted).

The text is categorized into various types such as for example

- Digit (e.g. '7' becomes 'seven')
- Cardinal (e.g. 211 becomes 'two hundred eleven')
- Ordinal ('1st' becomes 'first')
- Decimal (0.1 becomes 'point one')
- Fraction ('1/2' becomes 'one half')
- Money ($50 becomes 'fifty dollars')
- Measure ('6 mi' becomes 'six miles')
- Date (e.g. 2010 becomes 'twenty ten')
- Telephone ('911' becomes 'nine one one')
- Time ('8 a.m.' becomes 'eight a m')

The training dataset was downloaded from Kaggle competition web site has about 8 Million records across text types, with 133K records representing cardinal category. It was manually split into 50% training set (used to tune neural network parameters), 25% validation set (used to decide which network parameters to prefer) and 25% test set not used for modeling (only used to review model performs as expected.

Solution Statement

I would explore LSTM-based sequence-to-sequence neural network design that would translate input sequences consisting of symbols - to output sequences consisting of words or in more generality, symbols representing sounds that should be pronounced separately. Note that some translation words would be corresponding to one or more symbols (e.g. 11 would be translated to 'eleven', not 'one one'). This design would model on current work that typically includes encoder, decoder and possibly an attention mechanism, see detailed explanation below.

Benchmark Model

I would consider (likely human) translations provided as part of the modeling dataset - as primary benchmark. As a secondary benchmark, I consider to build a simpler feedforward (vanilla) neural network with up to 3 layers and appropriate number of hidden nodes, focused on generating translations one word at a time – so I could compare translation accuracy of my primary approach if it falls short of 100%. In situations when benchmark translation would disagree with the best model predictions, I would also consider manually reviewing machine translation for accuracy to review whether input data may be inaccurate.

As of writing of this plan, Kaggle leaderboard results seem to indicate that close to 100% accuracy may be achievable at least on the non-annotated dataset provided as test (not used in this project).

Evaluation Metrics

In this problem, I would aim for up to 100% translation accuracy on both development and test dataset, using solution provided in the competition. Accuracy would be defined as % of phrases correctly transcribed into spoken form. In case some instances would be incorrectly tagged, I could also visually review them for quality, augmenting data-driven evaluation with common sense review.

Project Design

I plan to implement a version of sequence-to-sequence neural machine translation architecture using supervised deep neural network fed with examples of translations – sequences as inputs and sequences as outputs[1]. Such architectures typically include:

- pre-processing of input and output sequences into vector representation
- encoder of input sequences into fixed-dimension vector (that is either being augmented at every step when new token is read) or sequence of vectors (separately output)
- step-by-step decoder that would generate machine translation tokens (words)
- possibly, attention mechanism that decoder would be using to decide which parts of the encoded input text to focus on

I plan to leverage one-hot encoding for pre-processing – symbol level for input and word level for translated (normalized) text.

Both encoder and decoded are typically setup as a stacked recurrent neural networks with long short-term memory (LSTM), for example

I would look for the network depth and dimensionality necessary to result in natural translation and manage-able training time.

To help decoder focus translation process, I intend to implement attention mechanism as simple one or two layers feed-forward neural network, executed at each step of generating output word to decide how to weigh different time steps of input. Using soft-max activation layer would represent output of this attention mechanism as probabilities (summing up to 1) with which different time steps should be weighted.

I would tokenize input text on a symbol-by-symbol level, and I would tokenize translated text into a word-by-word level to minimize the length of the translated sequence.

I would focus the translation on the digit / cardinal numeric data at first to ensure the process works and generates adequate translation. Once the process flow is established and ideally the system is able to learn sufficient structure on CPU, I may choose to scale the process to more complex forms of texts, and execute the machine learning process on the GPU such as amazon web services (AWS) cloud.

I would also like to visualize the translation process such as neural network architecture, data and attention flow - to better understand how the correct translation is performed.

[1] Sequences of translated text could have different length compared to text being translated