Project: Character Modeling

In this notebook I will use TensorFlow to create a Recurrent Neural Network (RNN), to predict the next character in a string. Training the network is performed using a CPU and using a GPU. The results are then compared to illustrate the performance of training the model using only a CPU versus using a GPU.

Table of Contents

Introduction

This code implements a Recurrent Neural Network with LSTM units for training/sampling from character-level language models. In other words, the model takes a text file as input and trains the RNN network that learns to predict the next character in a sequence.
The RNN can then be used to generate text character by character that will look like the original training data.

This code is based on this blog (http://karpathy.github.io/2015/05/21/rnn-effectiveness/), and the code is an step-by-step implementation of the character-level implimentation (https://github.com/crazydonkey200/tensorflow-char-rnn). Dataset can be downloaded from the following link (https://ibm.box.com/shared/static/a3f9e9mbpup09toq35ut7ke3l3lf03hg.txt).

The details about the project can be found on this online course (https://courses.edx.org/courses/course-v1:IBM+DL0122EN+3T2018/courseware/175796e419c1459da45ad967520dbe69/bf00b141dd5a4a148401f692fa8c7da7/1
activate_block_id=block-v1%3AIBM%2BDL0122EN%2B3T2018%2Btype%40vertical%2Bblock%404488372835844f4fb81b971b8e648291).

Model: Long Short-Term Memory Model (LSTM)

Recurrent Neural Networks are Deep Learning models with simple structures and a feedback mechanism built-in, or in different words, the output of a layer is added to the next input and fed back to the same layer.

The Recurrent Neural Network is a specialized type of Neural Network that solves the issue of **maintaining context for Sequential data** -- such as Weather data, Stocks, Genes, etc. At each iterative step, the processing unit takes in an input and the current state of the network, and produces an output and a new state that is **re-fed into the network**.

However, this model has some problems. It's very computationally expensive to maintain the state for a large amount of units, even more so over a long amount of time. Additionally, Recurrent Networks are very sensitive to changes in their parameters. To solve these problems, we use a specific type of RNN, is called Long Short-Term Memory (LSTM).

Each LSTM cell has 5 parts:

1. Input
2. prv_state
3. prv_output
4. new_state
5. new_output

- Each LSTM cell has an input layer, which its size is 128 units in our case. The input vector's dimension also is 128, which is the dimensionality of embedding vector, so called, dimension size of Word2Vec embedding, for each character.
- Each LSTM cell has a hidden layer, where there are some hidden units. The argument n_hidden=128 of BasicLSTMCell is the number of hidden units of the LSTM (inside A). It keeps the size of the output and state vector. It is also known as, rnn_size, num_units, num_hidden_units, and LSTM size, in literature.
- An LSTM keeps two pieces of information as it propagates through time:
  - **hidden state** vector: Each LSTM cell accept a vector, called **hidden state** vector, of size n_hidden=128, and its value is returned to the LSTM cell in the next step. The **hidden state** vector; which is the memory of the LSTM, accumulates using its (forget, input, and output) gates through time. "num_units" is equivalant to "size of RNN hidden state". Number of hidden units is the dimensianality of the output (= dimesianality of the state) of the LSTM cell.
  - **previous time-step output**: For each LSTM cell that we initialize, we need to supply a value (128 in this case) for the hidden dimension, or as some people like to call it, the number of units in the LSTM cell.

  </ul>

  Stacked LSTM

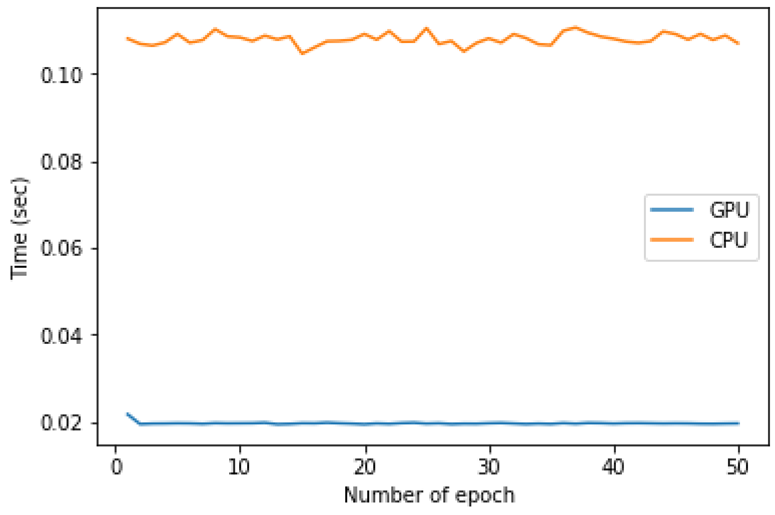
  What about if we want to have a RNN with stacked LSTM? For example, a 2-layer LSTM. In this case, the output of the first layer will become the input of the second.

  num_layers = 2

  - number of layers in the RNN, is defined by `num_layers` parameter.
  - An input of MultiRNNCell is **cells** which is list of RNNCells that will be composed in this order.

Results and conclusion

In this project, the character modeling was performed using a LSTM, a particular type of RNN using Tensorflow. TensorFlow's capability to execute the code on different devices such as CPUs and GPUs is a consequence of it's specific computational structure namely data flow graph. We have used this capability to our advantage e.g. in batch processing (parallelism). The result of parallel (GPUs) and serial (CPUs) processing is shown in the following figure. We can observe a 5 fold decrease in computational time with GPUs over CPUs. Therefore, training a deep learning model such as RNN over GPUs can be desirable for large scale computations.


Time (sec)
Number of epoch
GPU
CPU