METL TP4

Tientso Ning

Packaging and Development

1. We will use a virutal environment, then calling the command `python3 setup.py`. We use the virtual environment to isolate the packages and make sure that this does not interfere with our previous installations.

2. The `.yml` file is for Travis, and is used for continuous integration, specifically unit tests and code linting.

Running

1. The minimal command to train is `w2v train --data ./path_to_txt --outputdir ./output_path`

2. The training parameters are as follows:

- data: this specifies the data file that we need as input to train on (in our case, the wikipedia as txt file)
- outputdir: this specifies the output location where the trained model is stored
- alpha: the initial learning rate (we tune this to have it learn faster or slower)
- neg: the number of negative samples
- window: the window size
- epochs: the number of epochs (for training)
- size: the dimension of the vector
- min-count: the minimum frequency count
- sample: the subsampling rate (for training)
- train-mode: the type of training setup (i.e: we can do skipgram as shown in the default, or we can use cbow)
- t-num-threads: number of threads used for training (CPU stuff)
- p-num-threads: number of threads used for pre-processing (CPU stuff)
- keep-checkpoint-max: when training, there are certain checkpoints. This determines how many is kept during training (meaning if 0 or None, then there is no max and all are kept).
- batch: the batch size (for training)
- shuffling-buffer-size: the size of the buffer used for shuffling training data (for training)
- save-summary-steps: the number of steps after which summaries are saved.

- save-checkpoints-steps: the number of steps after which checkpoints are saved.
- log-step-count-steps: the number of global steps after which loss will be logged.

3. The default for each of the values are as follows:

- alpha: 0.025
- neg: 5
- window: 2
- epochs: 5
- size: 300
- min-count: 50
- sample: 1e-5
- train-mode: skipgram
- t-num-threads: 1
- p-num-threads: 1
- keep-checkpoint-max: 3
- batch: 1
- shuffling-buffer-size: 10000
- save-summary-steps: 100000
- save-checkpoints-steps: 1000000
- log-step-count-steps: 100000

It could be recommended to alter the number of threads used if your device can handle multiple threads (making training faster). Additionally, altering the batch size could also assist in efficiency.

Architecture

1. The architecture is as follows: split into pre-processing and training. Text -> preprocessing -> batch of tensors -> training -> word2vec vector representation.

2. The benefits of this architecture compared to the original tensorflow based estimator APIs. This high-level model abstraction allows you to run models locally using CPU/GPU without rewriting the models, and gives you a training loop that allows you to control when to load data, handle errors, create checkpoints, and have summaries easier.

Monitoring

1. The three categories under the SCALAR tab is:

- MEN: The pearson correlation computed on the MEN dataset.
- global_step: The number of global steps done per second.
- loss_1: This informs us of the loss of the model.

2. The regular performance drops under the global_step is most likely due to dumping the model to the disk. If we check the default hyper-parameters, we can see that the number of threads is set to 1.

3. Calculating the loss should take the most computation.

4. In this case, we would take the pre-processing task and give it to the CPU, and training to the GPU. We would then want to visualize the pre-processing stats for the CPU (which are not available in tensorboard) to ensure that pre-processing is not bottlenecking and causing the GPU to wait, which would effectively kill the efficiency and reason we parallelized.

Implementation

2. The original Word2vec implementation suggests using negative sampling, whereas we randomly choose a certain number of negative samples in order to update their weights. Whereas the implementation in TF uses Noise Constrastive Estimation, which sets up the problem in terms of sampling from a true distribution and a noise distribution (which we define). This allows us to draw the number of samples we want and to avoid having to deal with the entire vocabulary (which is big). The difference is that in NCE the probability that the sample comes from the noise-distribution is set based on the estimation done by a ratio of the sample coming from the true-distribution over the noise-distribution.

3. Our structure of the TF code requires the usage of higher order functions to wrap-around and provide information.