Sentiment Analysis Tutorial

Sentiment analysis has many applications. A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence or feature/aspect level. One simple example is to classify the customer reviews in a shopping website, a tourism website, and group buying websites like Amazon, TaoBao, Tmall etc.

Sentiment analysis is also used to monitor social media based on large amount of reviews or blogs. For example, the researchers analyzed several surveys on consumer confidence and political opinion, found they correlate to sentiment word frequencies in contemporaneous Twitter messages [1]. Another example is to forecast stock movements through analyzing the text content of a daily Twitter blog [2].

On the other hand, grabbing the user comments of products and analyzing their sentiment are useful to understand user preferences for companies, products, even competing products.

This tutorial will guide you through the process of training a Long Short Term Memory (LSTM) Network to classify the sentiment of sentences from Large Movie Review Dataset, sometimes known as the Internet Movie Database (IMDB). This dataset contains movie reviews along with their associated binary sentiment polarity labels, namely positive and negative. So randomly guessing yields 50% accuracy.

Data Preparation

IMDB Data Introduction

Before training models, we need to preprocess the data and build a dictionary. First, you can use following script to download IMDB dataset and Moses tool, which is a statistical machine translation system. We provide a data preprocessing script, which is capable of handling not only IMDB data, but also other user-defined data. In order to use the pre-written script, it needs to move labeled train and test samples to another path, which has been done in `get_imdb.sh`.

```
cd demo/sentiment/data
./get_imdb.sh
```

If the data is obtained successfuly, you will see the following files at `./demo/sentiment/data`:

```
aclImdb  get_imdb.sh  imdb  mosesdecoder-master
```

- aclImdb: raw dataset downloaded from website.
- imdb: only contains train and test data.
- mosesdecoder-master: Moses tool.

IMDB dataset contains 25,000 highly polar movie reviews for training, and 25,000 for testing. A negative review has a score ⩽ 4 out of 10, and a positive review has a score ⩾ 7 out of 10. After running `./get_imdb.sh`, we can find the dataset has the following structure in `aclImdb`.

```
imdbEr.txt  imdb.vocab  README  test  train
```

- train: train sets.
- test : test sets.
- imdb.vocab: dictionary.
- imdbEr.txt: expected rating for each token in imdb.vocab.
- README: data documentation.

Both train and test set directory contains:

```
labeledBow.feat  neg  pos  unsup  unsupBow.feat  urls_neg.txt  urls_pos.txt  urls_unsup
```

- pos: positive samples, contains 12,500 txt files, each file is one movie review.
- neg: negative samples, contains 12,500 txt files, each file is one movie review.
- unsup: unlabeled samples, contains 50,000 txt files.
- urls_xx.txt: urls of each reviews.
- xxBow.feat: already-tokenized bag of words (BoW) features.

IMDB Data Preparation

In this demo, we only use labled train and test set and not use imdb.vocab as dictionary. By default, dictionary is builded on train set. Train set is shuffled and test set is not. `tokenizer.perl` in Moses tool is used to tokenize the words and punctuation. Simply execute the following command to preprcess data.

```
cd demo/sentiment/
./preprocess.sh
```

preprocess.sh:

```
data_dir="./data/imdb"
python preprocess.py -i data_dir
```

- data_dir: input data directory.
- preprocess.py: preprocess script.

If running successfully, you will see `demo/sentiment/data/pre-imdb` directory as follows:

```
dict.txt  labels.list  test.list  test_part_000  train.list  train_part_000
```

- test_part_000 and train_part_000: all labeled test and train sets. Train sets have be shuffled.
- train.list and test.list: train and test file lists.
- dict.txt: dictionary generated on train sets by default.
- labels.txt: neg 0, pos 1, means label 0 is negative review, label 1 is positive review.

User-defined Data Preparation

If you perform other sentiment classifcation task, you can prepare data as follows. We have provided the scripts to build dictionary and preprocess data. So just organize data as follows.

```
dataset
|----train
|     |----class1
|     |     |----text_files
|     |----class2
|     |     |----text_files
|     |     ...
|----test
|     |----class1
|     |     |----text_files
|     |----class2
|     |     |----text_files
|     |     ...
```

- dataset: 1st directory.
- train, test: 2nd directory.
- class1,class2,...: 3rd directory.
- text_files: samples with text file format.

All samples with text files format under the same folder are same category. Each text file contains one or more samples and each line is one sample. In order to shuffle fully, the preprocessing is a little different for data with multiple lines in one text file, which needs to set `-m True` in `preprocess.sh`. And tokenizer.perl is used by default. If you don't need it, only set `-t False` in `preprocess.sh'.

Training

In this task, we use Recurrent Neural Network (RNN) of LSTM architecure to train sentiment analysis model. LSTM model was introduced primarily in order to overcome the problem of vanishing gradients. LSTM network resembles a standard recurrent neural network with a hidden layer, but each ordinary node in the hidden layer is replaced by a memory cell. Each memory cell contains four main elements: an input gate, a neuron with a self–recurrent connection, a forget gate and an output gate. More details can be found in the literature [4]. The biggest advantage of the LSTM architecture is that it learns to memorize information over long time intervals without the loss of short time memory. At each time step with a new coming word, historical information stored in the memory block is updated to iteratively learn the sequence representation.

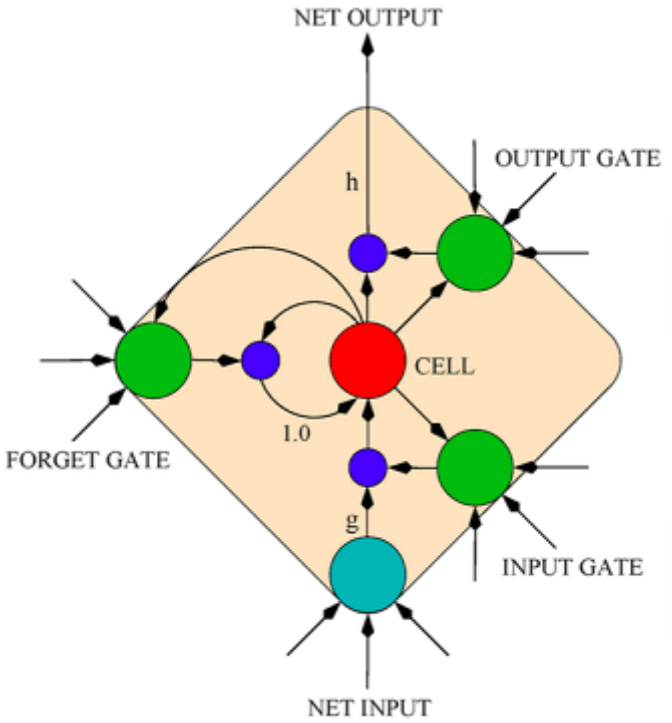

Figure 1. LSTM [3]

Sentiment analysis is among the most typical problems in natural language understanding. It aims at predicting the attitude expressed in a sequence. Usually, only some key words, like adjectives and adverbs words, play a major role in predicting the sentiment of sequences or paragraphs. However, some review or comment contexts are very long, such as IMDB dataset. We use LSTM to perform this task for its improved design with the gate mechanism. First, it is able to summarize the representation from word level to context level with variable context length which is adapted by the gate values. Second, it can utilize the expanded context at the sentence level, while most methods are good at utilizing n–gram level knowledge. Third, it learns the paragraph representation directly rather than combining the context level information. This results in this end–to–end framework.

In this demo we provide two network, namely bidirectional–LSTM and three layers of stacked–LSTM.

Bidirectional-LSTM

One is a bidirectional LSTM network, connected by fully connected layer and softmax, as shown in Figure 2.

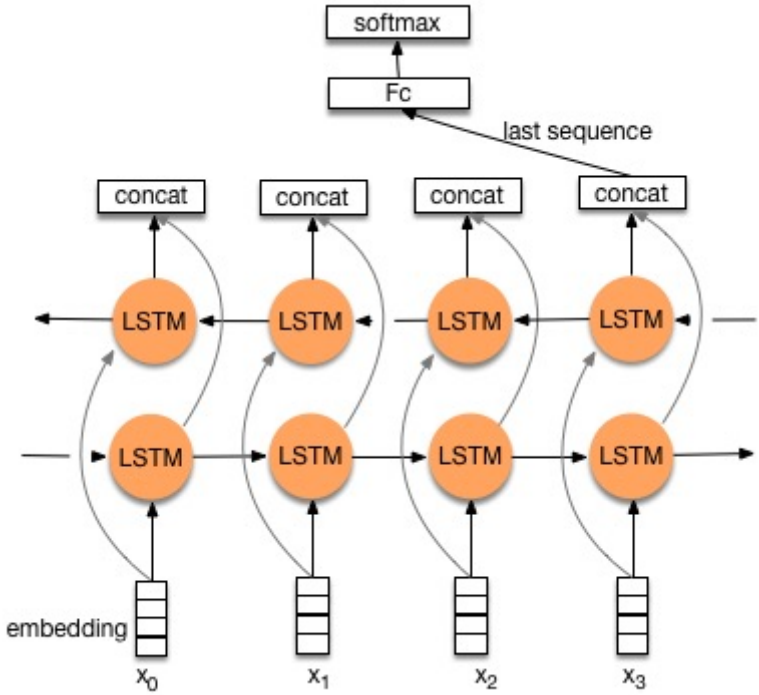

Figure 2. Bidirectional–LSTM

Stacked-LSTM

Another is three–layer LSTM structure in Figure 3. The bottom of the figure is word embedding. Next, three LSTM–Hidden layers are connected and the second LSTM is reversed. Then extract the maximum hidden vectors of all time step of hidden and LSTM layer as the representation for the entire sequence. Finally, a fully connected feed forward layer with softmax activation is used to perform the classification task. This network is refered to paper [5].

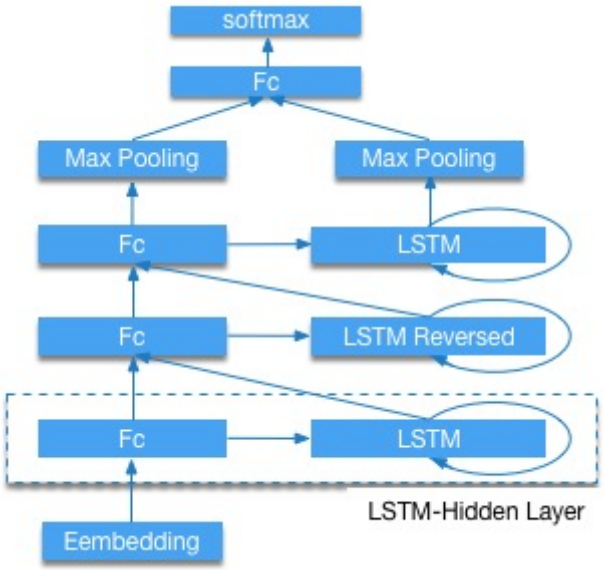

Figure 3. Stacked–LSTM for sentiment analysis

Config

Switch into `demo/sentiment` directory, `trainer_config.py` file is an example of the config, containing algorithm and newtork configure. The first line imports predefined networks from `sentiment_net.py`.

trainer_config.py:

```
from sentiment_net import *

data_dir  = "./data/pre-imdb"
# whether this config is used for test
is_test = get_config_arg('is_test', bool, False)
# whether this config is used for prediction
is_predict = get_config_arg('is_predict', bool, False)
dict_dim, class_dim = sentiment_data(data_dir, is_test, is_predict)

################## Algorithm Config #####################

settings(
  batch_size=128,
  learning_rate=2e-3,
  learning_method=AdamOptimizer(),
  regularization=L2Regularization(8e-4),
  gradient_clipping_threshold=25
)

#################### Network Config ######################
stacked_lstm_net(dict_dim, class_dim=class_dim,
                 stacked_num=3, is_predict=is_predict)
#bidirectional_lstm_net(dict_dim, class_dim=class_dim, is_predict=is_predict)
```

- **Data Definition**:
  - get_config_arg(): get arguments setted by `--config_args=xx` in commandline argument.
  - Define TrainData and TestData provider, here using Python interface (PyDataProviderWrapper) of PaddlePaddle to load data. For details, you can refer to the document of PyDataProvider.
- **Algorithm Configuration**:
  - use sgd algorithm.
  - use adam optimization.
  - set batch size of 128.
  - set average sgd window.
  - set global learning rate.
- **Network Configuration**:
  - dict_dim: get dictionary dimension.
  - class_dim: set category number, IMDB has two label, namely positive and negative label.
  - `stacked_lstm_net`: predefined network as shown in Figure 3, use this network by default.
  - `bidirectional_lstm_net`: predefined network as shown in Figure 2.

Training

Install PaddlePaddle first if necessary. Then you can use script `train.sh` as follows to launch local training.

```
cd demo/sentiment/
./train.sh
```

train.sh:

```
config=trainer_config.py
output=./model_output
```

```
paddle train --config=$config \
             --save_dir=$output \
             --job=train \
             --use_gpu=false \
             --trainer_count=4 \
             --num_passes=10 \
             --log_period=20 \
             --dot_period=20 \
             --show_parameter_stats_period=100 \
             --test_all_data_in_one_period=1 \
             2>&1 | tee 'train.log'
```

- --config=$config: set network config.
- --save_dir=$output: set output path to save models.
- --job=train: set job mode to train.
- --use_gpu=false: use CPU to train, set true, if you install GPU version of PaddlePaddle and want to use GPU to train.
- --trainer_count=4: set thread number (or GPU count).
- --num_passes=15: set pass number, one pass in PaddlePaddle means training all samples in dataset one time.
- --log_period=20: print log every 20 batches.
- --show_parameter_stats_period=100: show parameter statistic every 100 batches.
- --test_all_data_in_one_period=1: test all data every testing.

If the run succeeds, the output log is saved in path of `demo/sentiment/train.log` and model is saved in path of `demo/sentiment/model_output/`. The output log is explained as follows.

```
Batch=20 samples=2560 AvgCost=0.681644 CurrentCost=0.681644 Eval: classification_error_
...
Pass=0 Batch=196 samples=25000 AvgCost=0.418964 Eval: classification_error_evaluator=0.
Test samples=24999 cost=0.39297 Eval: classification_error_evaluator=0.149406
```

- Batch=xx: means passing xx batches.
- samples=xx: means passing xx samples.
- AvgCost=xx: averaged cost from 0-th batch to current batch.
- CurrentCost=xx: current cost of latest log_period batches.
- Eval: classification_error_evaluator=xx: means classfication error from 0-th batch ro current batch.
- CurrentEval: classification_error_evaluator: current classfication error of the lates log_period batches.
- Pass=0: Going through all training set one time is called one pass. 0 means going through training set first time.

By default, we use the `stacked_lstm_net` network, which converges at a faster rate than `bidirectional_lstm_net` when passing same sample number. If you want to use bidirectional LSTM, just remove comment in the last line and comment `stacked_lstm_net`.

Testing

Testing means evaluating the labeled validation set using trained model.

```
cd demo/sentiment
./test.sh
```

test.sh:

```
function get_best_pass() {
  cat $1  | grep -Pzo 'Test .*\n.*pass-.*' | \
  sed  -r 'N;s/Test.* error=([0-9]+\.[0-9]+).*\n.*pass-([0-9]+)/\1 \2/g' | \
  sort | head -n 1
}

log=train.log
LOG=`get_best_pass $log`
LOG=(${LOG})
evaluate_pass="model_output/pass-${LOG[1]}"

echo 'evaluating from pass '$evaluate_pass

model_list=./model.list
touch $model_list | echo $evaluate_pass > $model_list
net_conf=trainer_config.py
paddle train --config=$net_conf \
             --model_list=$model_list \
             --job=test \
             --use_gpu=false \
             --trainer_count=4 \
             --config_args=is_test=1 \
             2>&1 | tee 'test.log'
```

The function `get_best_pass` gets the best model by classification error rate for testing. In this example, We use test dataset of IMDB as validation by default. Unlike training, it needs to specify `--job=test` and model path, namely `--model_list=$model_list` here. If running successfully, the log is saved in path of `demo/sentiment/test.log`. For example, in our test, the best model is `model_output/pass-00002`, the classification error is 0.115645 as follows.

```
Pass=0 samples=24999 AvgCost=0.280471 Eval: classification_error_evaluator=0.115645
```

Prediction

`predict.py` provides a predicting interface. You should install python api of PaddlePaddle before using it. One example to predict unlabeled review of IMDB is as follows. Simply running:

```
cd demo/sentiment
./predict.sh
```

predict.sh:

```
#Note the default model is pass-00002, you shold make sure the model path
#exists or change the mode path.
model=model_output/pass-00002/
config=trainer_config.py
label=data/pre-imdb/labels.list
python predict.py \
     -n $config\
     -w $model \
     -b $label \
     -d data/pre-imdb/dict.txt \
     -i data/aclImdb/test/pos/10007_10.txt
```

- `predict.py`: predicting interface.
- –n $config : set network configure.
- –w $model: set model path.

- –b $label: set dictionary about corresponding relation between integer label and string label.
- –d data/pre–imdb/dict.txt: set dictionary.
- –i data/aclImdb/test/pos/10014_7.txt: set one example file to predict.

Note you should make sure the default model path `model_output/pass-00002` exists or change the model path.

Predicting result of this example:

```
Loading parameters from model_output/pass-00002/
./data/aclImdb/test/pos/10014_7.txt: predicting label is pos
```

We sincerely appreciate your interest and welcome your contributions.

Reference

[1] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In ICWSM–2010.
[2] Johan Bollen, Huina Mao, Xiaojun Zeng. 2011. Twitter mood predicts the stock market, Journal of Computational Science.
[3] Alex Graves, Marcus Liwicki, Santiago Fernan– dez, Roman Bertolami, Horst Bunke, and Ju ̈rgen Schmidhuber. 2009. A novel connectionist system for unconstrained handwriting recognition. IEEE Transactions on Pattern Analysis and Machine In– telligence, 31(5):855—868.
[4] Zachary C. Lipton, A Critical Review of Recurrent Neural Networks for Sequence Learning, arXiv:1506.00019.
[5] Jie Zhou and Wei Xu; End–to–end Learning of Semantic Role Labeling Using Recurrent Neural Networks; ACL–IJCNLP 2015.