Learning Continuous Phrase Repesentations for Translation Modelling

Course: Machine Learning and Translation

Kerry Zhang 2014403073, Richard Luong 2014403075

January 29, 2015

Contents

1 Distribution of work between the authors

To be sure that both of us made equal amount of work, the Pair programming [4] technique was applied.

2 Description

Our project is to implement Phrase Embedding, as described in the paper by Gao et al [1].

So when the system is trained, it will output a real number vector given a source sentence. For two similar sentence, the output differences should be smaller, than for two completely different ones.

This is done by a neural network, with one hidden layer. The first layer has the same number of nodes as the vocabulary size of both lanugages. The second and the third layer is set to 100 nodes.

The initialization of the weights of the first layer to the hidden layer, denoted as **W1**, is done by using a bilingual topic distribution, in this case solved by using Latent Dirchlet Allocation. The weights between the hidden layer and the third layer, **W2**, is an identity matrix.

Optimization of the weights are done with Stochastic gradient descent [5]. The training should be stopped according to early stop principle, as suggested in the paper, however we haven't fine tuned the system sufficiently to determine a stop condition as of yet.

3 Dependencies

- python 2.7+
- NumPy [2]
- Gensim [6]

4 How to run

Unzip the archive.

Before the first run, **W1** needs to be initialized, using Latent Dirchlet Allocation [3].

```
python preprocessing.py corpus_source corpus_target n_best_list
    reference_file
```

where `corpus_source` and `corpus_target` are paths to the parallell corpuses the bilingual topic modelling should be based on, `n_best_list` is a path to a list of the n best translation of source sentences and `reference_file` is a path to the reference translation of those source sentences. Sentence-level BLEU score is also computed during this stage. The W1 initialization and BLEU score can be found in `data/weight_initialization.gz` and `sbleu.txt`

After that the system should be trained with

```
python main.py $source_sentence_file $n_best_list
```

where `$source` is a path to the source sentences and `$n_best_list` is a path to the n best translation of these sentences.

5 Output

When the system converges, the weights of the neural network are written to the files `W1.gz` and `W2.gz`.

6 Experiment Results and Discussion

After multiple overnight test runs of the system that resulted in calculations that outputted NaN due to calculations that accumulated into too large or small numbers, and other issues such as too big gradients resulting in a very unstable gradient descent, we finally managed to get a relatively stable system. The two main parameters that required tuning were: `learning_rate` and `smoothing_factor`, which can be altered at the top of `main.py`. See Equation 6 in the paper by Gao et. al. [1] for an explanation of the smoothing factor.

As of now, the most stable system we have produced came when the parameters were set as follows: `smoothing_factor = 10` and `learning_rate = 1000`. With these parameter values, we conducted two overnight test runs.

The first was to test the theoretical validity of our system, i.e. whether or not the implemented calculations of the gradients, as suggested by the paper, actually result in a lowering of the loss function defined as $-xBleu$. To do this, we used 30 training samples (sentence number 200 to 230 in the file `test.input.tok.` provided by our teacher), performed the gradient descent training on the entire training set and then tested the new average $-xBleu$ score for the entire training set. I.e. we used the training set as our test set in order to test the theoretical validity of our neural network. The results (See Figure 6) seem to suggest that the system is indeed performing a correct gradient descent.

The second overnight experiment was conducted on a larger training set with 180 training samples and 20 test samples, this time using source sentences 1 to 180 from our input file `test.input.tok.1` as training samples and sentences 181 to 200 as test samples. This choice of training samples means that the result are independent of the result in the first test, as none of the training samples overlap. Again, the result of our experiment suggest that the system works (see Figure 6), however after 14 epochs the gradients became very large and the system calculations outputted NaN again. Detailed system output can be found in the `lcprtm/Result` directory and the results discussed in this section can be found in the `lcprtm/Result/csv/` directory (also included in Appendix A.1 and A.2).

Figure 1: The results fr the first overnight experiment.

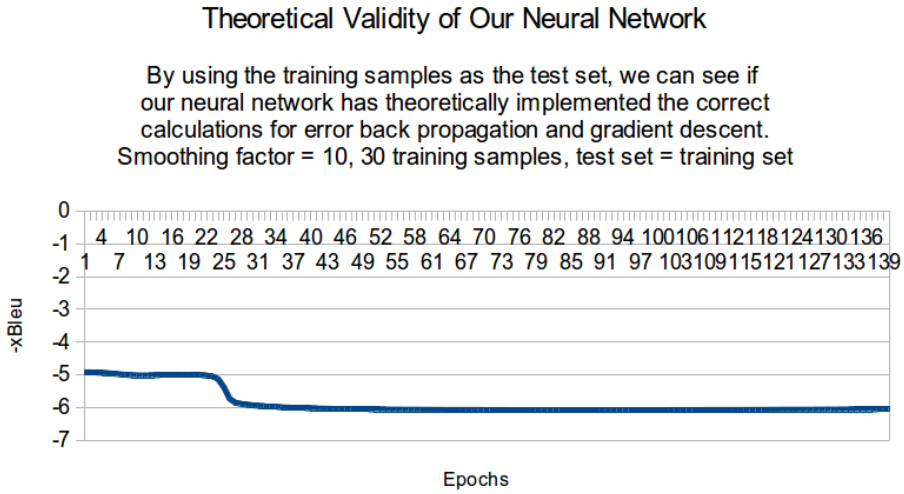
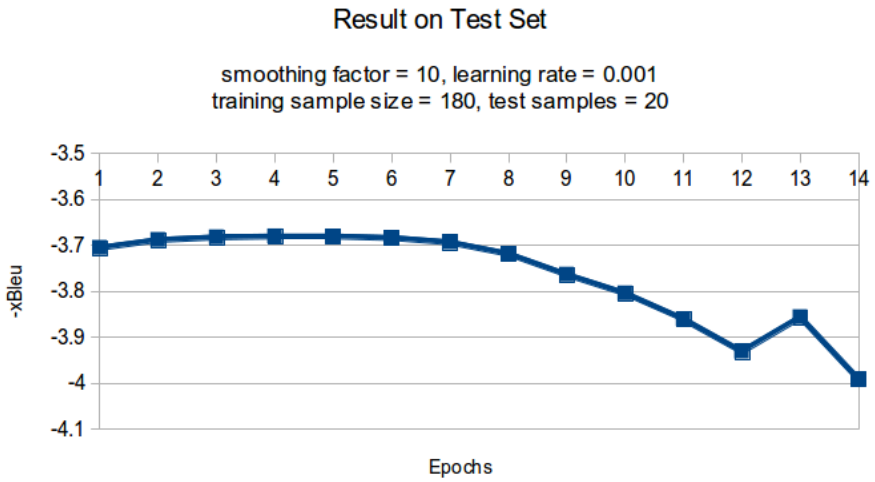


Figure 2: The results for second overnight experiment.



7 Conclusion

We believe that our neural network correctly learns how to minimize the loss function $-xBleu$ and is relatively stable. Increasing `smoothing_factor` will result in smaller gradients and slower training. Because we are getting NaN values (too big or too small numbers) after certain amounts of epochs with large training sets, we believe that `smoothing_factor` should be increased for a more stable system. By how much can only be determined by trial and error, more experiments need to be conducted in order to tune the parameter `smoothing_factor` in particular, and possibly `learning_rate`.

As of now, the training will run forever as we have not been able to tune the parameters for a sufficiently stable system in order to determine a good convergence criteria.

References

[1] Jianfeng Gao, Xiaodong He, Wen-tau Yih, and Li Deng. Learning continuous phrase representations for translation modeling. In *Proc. ACL*, 2014.

[2] NumPy. Numpy - numpy. `http://www.numpy.org/`, 2015. [Online; accessed 28-January-2015].

[3] Wikipedia. Latent dirichlet allocation — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Latent_Dirichlet_allocation&oldid=644441310`, 2015. [Online; accessed 28-January-2015].

[4] Wikipedia. Pair programming — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Pair_programming&oldid=642184088`, 2015. [Online; accessed 28-January-2015].

[5] Wikipedia. Stochastic gradient descent — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=642575072`, 2015. [Online; accessed 28-January-2015].

[6] Radim ehek. gensim: Topic modelling for humans. `https://radimrehurek.com/gensim/index.html`, 2015. [Online; accessed 28-January-2015].

A Appendix

A.1 Results for first run

Loss function
-3.705635097902438169
-3.688438543285909699
-3.682763400473420656
-3.680557770067027779
-3.680711280606880909
-3.684001256719885653
-3.693780268541727363
-3.718753361569778981
-3.763925242035965457
-3.804602229567417560
-3.860868119753573691
-3.931406370914172221
-3.856483504073254576
-3.991880703457518997

Table 1: Results for first run.

A.2 Results for second run

Loss function	Loss function (2)	Loss function (3)
-4.925219768943486898	-6.053793053843013361	-6.083056022301278176
-4.928882439772602098	-6.056059390669773279	-6.082919192595031888
-4.934277387580417340	-6.058154346998018802	-6.082761103000437153
-4.942461536588248272	-6.060093667926703453	-6.082581753490504006
-4.953402019294050440	-6.061891273652981305	-6.082381137947021621
-4.967205169862979552	-6.063559508517056607	-6.082159247703573079
-4.983938877082958463	-6.065109350617155393	-6.081916074959402785
-5.001246771383204504	-6.066550589070367927	-6.081651616059136067
-5.014515411532945777	-6.067891974564887825	-6.081365874633889668
-5.020956589315610508	-6.069141347741340731	-6.081058864599810931
-5.021672773425295055	-6.070305749074819346	-6.080730613011120944
-5.019218124702288009	-6.071391513247576199	-6.080381162765855940
-5.015542066372502994	-6.072404350461538591	-6.080010575163440123
-5.011747633699355831	-6.073349416708183846	-6.079618932314816426
-5.008397666305318907	-6.074231374666536176	-6.079206339406809434
-5.005783489490135629	-6.075054446619877702	-6.078772926823822864
-5.004103627415170230	-6.075822460554236137	-6.078318852131030958
-5.003577281610437133	-6.076538890415599248	-6.077844301924228532
-5.004508825520574078	-6.077206891350543927	-6.077349493552406834
-5.007374718842890537	-6.077829330628897964	-6.076834676719656159
-5.013181137544878041	-6.078408814842939378	-6.076300134973541489
-5.024749137618814032	-6.078947713890874738	-6.075746187087095684
-5.051203953979976902	-6.079448182179837623	-6.075173188341578268
-5.126880695576494062	-6.079912177422676756	-6.074581531716643390
-5.377137947495798542	-6.080341477351625556	-6.073971648993846451
-5.738500425794431870	-6.080737694628298939	-6.073344011778441853
-5.858067110992541515	-6.081102290192462512	-6.072699132443033498
-5.893931351373623784	-6.081436585260475702	-6.072037564995075520
-5.917728160083535194	-6.081741772156892267	-6.071359905868195739
-5.936434754999877939	-6.082018924139373617	-6.070667794635203090
-5.951938120110934705	-6.082269004356468223	-6.069958914638178271
-5.965122765282610651	-6.082492874060158705	-6.069236993528223323
-5.976512393355510433	-6.082691300179127403	-6.068501803704688058
-5.986458646403547412	-6.082864962345139404	-6.067754162640379256
-5.995216147764370618	-6.083014459452415146	-6.066994933076196084
-6.002978262580414004	-6.083140315818993393	-6.066225023065030975
-6.009896557928029459	-6.083242987009279901	-6.065445385841475634
-6.016092512769798617	-6.083322865368059773	-6.064657019490308443
-6.021665115471935614	-6.083380285308422231	-6.063860966383263218
-6.026696076217249676	-6.083415528388702143	-6.063058312350388945
-6.031253554527745031	-6.083428828207362749	-6.062250185548935910
-6.035394909507702188	-6.083420375138604008	-6.061437754990152804
-6.039168778461275977	-6.083390320926705286	-6.060622228681596724
-6.042616678100265482	-6.083338783152148821	-6.059804851340848231
-6.045774257236814719	-6.083265849578796569	-6.058986901635067035
-6.048672289554978043	-6.083171582387856269	-6.058169688900330030
-6.051337469104096201		-6.057354549295030033

Table 2: Results for second run.