MosQEto: Machine Translation Quality Estimation Framework

Vilém Zouhar
vilem.zouhar@mff.cuni.cz

Ondřej Měkota
o@mkta.eu

January 19, 2021

Abstract

Word level quality estimation (QE) of machine translation is a task aiming to identify badly translated words and spaces between words. We propose a framework for experiment replication of QE systems MosQEto. We were also experimenting with several methods, trying to improve the quality estimation of systems implemented in OpenKiwi by smartly preprocessing and synthesizing training data.

***K*eywords** Quality Estimation · Machine Translation · WMT

1 Introduction

Word level quality estimation is a task that receives the source of the translation and the translation itself and outputs which words in the output are translated correctly, usually in terms of probabilities. In WMT (since 2012) this is made into a classification task of every target token being either `OK` or `BAD`.

QE can be either supervised or unsupervised. The supervised version also gets the correct classification tags for each word and space between words during training. The unsupervised approaches rely mostly on getting self-reported confidence from machine translation models themselves. To our knowledge, this approach is much less reliable.

There are also document-level, paragraph-level, and sentence-level QE tasks. However, we did not consider those, because there are already ways of inferring higher-level estimations from word-level. We would also like to cast a shadow of doubt on this approach. It can often happen that every target word is part of a good translation, but the sentence as a whole is a bad translation. This can be supported by the fact that in BLEU, one can see examples of translations with very high bigram precision and very low or zero quadgram precision. From this perspective, sentence-level makes more sense. For sentence-level, HTER is used, which measures the amount of post-editing work needed to produce a good translation.

QE is becoming popular in translation companies for estimating post-editing costs.

2 Word-level Quality Estimation

In fig. 1 there is an illustration of word-level QE input, and output. Part of the QE task is to also predict missing words.

There are very few `BAD` tags, most word are translated correctly. Because of this imbalance F_1 is used for QE system evaluation. F_1 is computed separately for both `OK` tags (those that are classified as correctly translated) and `BAD` (those that are classified as incorrectly translated). These two computed numbers are then multiplied together yielding a $F_{MULTI} = F_{BAD} \cdot F_{OK}$ metric. Using F_{MULTI} forbids cheating by e.g. only prediction `OK`, because F_{OK} would be high, but F_{BAD} zero.

There are other possibilities of evaluation metrics, like mean square error, Spearman's, or Person's correlation, but they are used mostly for higher-level QE tasks.

The state of the art results for NMT QE models estimating for NMT MT models is 47.5% for English-German [Fonseca et al., 2019].

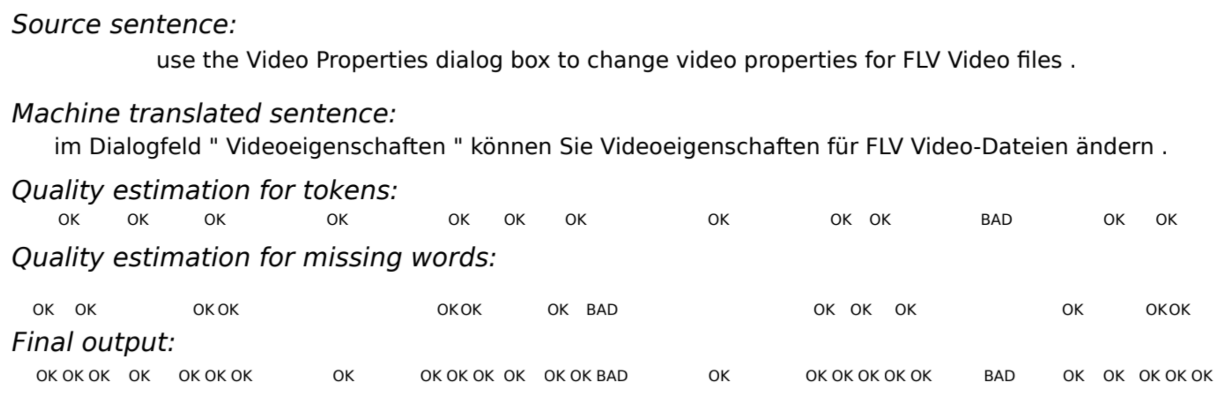
Source sentence:
use the Video Properties dialog box to change video properties for FLV Video files .
Machine translated sentence:
im Dialogfeld " Videoeigenschaften " können Sie Videoeigenschaften für FLV Video-Dateien ändern .
Quality estimation for tokens:
OK OK OK OK OK OK OK OK OK OK BAD OK OK
Quality estimation for missing words:
OK OK OK OK OK OK OK BAD OK OK OK OK OK OK
Final output:
OK OK OK OK OK OK OK OK OK OK OK OK OK OK BAD OK OK OK OK OK OK BAD OK OK OK OK OK

Figure 1: Quality estimation example.

3 Quality Estimation Systems

There are several architectures and tools to perform quality estimation: QuEst++ [Specia et al., 2015], DeepQuest [Ive et al., 2018] and OpenKiwi [Kepler et al., 2019].

We have used OpenKiwi to perform our experiments because of the easy integration into our codebase. OpenKiwi is an open-source tool and contains four different architectures to do QE. We used only two: NuQE and QUETCH. This was so mainly because we did not aim to achieve the highest QE performance but only to compare our data preprocessing and synthesis tool with the baseline. PredEst and biRNN, the other models in OpenKiwi, take significantly more time to train.

QUETCH First system that we used is a simple two layer neural network, the first layer has dimensionality 300 and the second 50. It is fed word embeddings, of dimension 50 in our case, and it outputs probability distribution over `OK`/`BAD` tags.

NuQE On top of four densely connected layer, NuQE contains also bi-directional GRU layer. Because of the GRU layer, NuQE takes significantly more time to train compared to QUETCH.

4 MosQEto

MosQEto is an open source[1] framework which allows researches to perform experiments defined by a simple YAML file. This YAML file represents a set of data and a processing pipeline. The pipeline can contain functions that process the data and perform training, testing, printing debugging messages.

The whole project is also integrated with fast_align [Dyer et al., 2013] so that word alignment can be easily generated without any additional work from the experiment designer.

```
dataset_load:
  - "opus/tech/en-de"
  - "wmt19/en-de"

dataset_save:
  - "data_all/"

method:
  - Loader:rmdir:data_tmp
  - Loader:load:noalign
  - DataUtils:info
  - Generator:synthetize_random
  - DataUtils:head:7000
  - DataUtils:info

  - Loader:load
  - DataUtils:info

  - Generator:mix
  - Loader:save

generator:
  change_prob:  0.12
```

Listing 1: Example config file for data synthesis in MosQEto

The example configuration file in listing 1 has a stack of two datasets: WMT19 QE annotated data and OPUS IT domain [Tiedemann, 2012]. They are loaded in order by calling `Loader:load` (each call loads one dataset into the buffer). QE data is synthesized from the OPUS dataset, cropped to 7000 units, and mixed with WMT19 QE data.

4.1 Commands

Here we list the available commands in MosQEto experiment definitions.

- `Loader:rmdir:D` removes the `D` directory. In MosQEto we used `data_tmp` for intermediate results storage.
- `Loader:load` loads a new dataset from the top of the stack. Optionally `:noalign` can be specified to skip fast_align invocation as it takes quite some time to finish. The corpora are loaded and downloaded dynamically. Thus, the user of MosQEto does not have to go through the process of manually downloading and unpacking datasets.
- `Loader:save` saves the current working data to a directory contained at the top of the save stack.
- `DataUtils:info` prints basic statistics about the data such as the distribution of `OK` and `BAD` tags. This command was crucial to us for data analysis.
- `DataUtils:flush_train` removes the `train` part from the dataset.
- `DataUtils:split_dev:P` moves P% of the `train` units to `dev` part.
- `DataUtils:head:N` takes the first `N` units from the top of the dataset.
- `Generator:synthetize_random` acts according to the `generator` object which can contain `change_prob`, `remove_prob` and `append_prob` for change, removal or insertion of words respectively. It takes every pair of

[1]Contributions, and usage reports welcome at github.com/zouharvi/MosQEto

(sentence, translation) and executes the given operation. For example in listing 1 it changes each target word with the probability 12% and sets the corresponding tag to `BAD`. Other tags are set to `OK`.

- `Generator:expand_post_edited` simply takes all the post-edited sentences and sets their tags as `OK`.
- `Generator:synthetize_gold` Marks every target token as `OK` or `BAD` based on whether it appears in the post-edited text. Optionally if the `generator` object has `case_insensitive` set to `True` then the equivalency is compared without case sensitivity.
- `Generator:mix` shuffles the dataset.

5 Experiments

We experiment with two datasets: the WMT19 Quality Estimation dataset (IT domain) and the OPUS from Ubuntu, PHP, KDE4, and Gnome [Tiedemann, 2012]. WMT19 contains 14 thousand parallel sentences and also post-edited sentences. Our part of OPUS on which we focused contains 300 thousand parallel sentences.

We experimented with dataset transformations and with transfer learning. All the experiments are documented[2] in our GitHub repository together with the experiment definitions[3] which can be used to reproduce our results.

Baseline Our baseline is training QUETCH for 30 epochs, the $F1_{MULTI}$ is 28.8% and none of our experiments surpassed it. We used Adam optimizer [Kingma and Ba, 2014] with learning rate 10^{-4}, 0.1 dropout, and batch size 64.

Small OPUS only with changes We also trained a model on the OPUS dataset but preprocessing it such that for each word, a tag was switched to "BAD" with 12% probability. The F1 score was 16%.

WMT19 + WMT19 post-edited The next experiment was training on WMT19 extended by the post-edited sentences, all marked "OK". This, however, did not improve the F1 score: 25.1%.

WMT19 + Small OPUS only with changes Another experiment combined WMT19 data with those OPUS preprocessed data (with 12% tag change to "BAD"). This setup matched the baseline performance, 28.8%.

WMT19 + PE synth WMT19 was enriched by WMT19 data with different tags: a word is marked "OK" if and only if the word also appears in the post-edited version, "BAD" otherwise. This results in 25.9% F1 score.

Transfer Learning The transfer learning experiment failed badly. We suspect it is due to an error in the OpenKiwi framework. We trained for 30 epochs on OPUS, and then for another 30 epochs on WMT19 and evaluated on WMT19. The F1 was only 7%, which is 4 times lower than the baseline. This was after extensive hyperparameter search and even after using NuQE instead of QUETCH. We tried to identify the error by trying to train on WMT19, save the model, and train again on WMT19, but there was still a drop in F1.

6 Conclusion

Even though we did not achieve any improvements over the baseline, we managed to implement (most of our dev time spent here) a robust system for experiment replication. An experiment of any kind (training, testing, data creation) can be easily defined using a single YAML file. MosQEto is oriented for data manipulation, but it can be used for anything if relevant `Worker` classes are implemented.

In our future work we would like to revise what went wrong with the transfer learning and come up with new ways of QE data synthesis. We mostly hoped for the post-edited data synthesis, but that failed.

[2]github.com/zouharvi/MosQEto/blob/master/docs/experiments.md
[3]github.com/zouharvi/MosQEto/tree/master/config

References

[Dyer et al., 2013] Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of NAACL-HLT*, pages 644–648.

[Fonseca et al., 2019] Fonseca, E., Yankovskaya, L., Martins, A. F., Fishel, M., and Federmann, C. (2019). Findings of the wmt 2019 shared tasks on quality estimation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 1–10.

[Ive et al., 2018] Ive, J., Blain, F., and Specia, L. (2018). Deepquest: a framework for neural-based quality estimation. *In the Proceedings of COLING 2018, the 27th International Conference on Computational Linguistics, Sante Fe, New Mexico, USA*.

[Kepler et al., 2019] Kepler, F., Trénous, J., Treviso, M., Vera, M., and Martins, A. F. T. (2019). OpenKiwi: An open source framework for quality estimation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics–System Demonstrations*, pages 117–122, Florence, Italy. Association for Computational Linguistics.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

[Specia et al., 2015] Specia, L., Paetzold, G., and Scarton, C. (2015). Multi-level translation quality prediction with quest++. In *ACL-IJCNLP 2015 System Demonstrations*, pages 115–120, Beijing, China.

[Tiedemann, 2012] Tiedemann, J. (2012). Parallel data, tools and interfaces in opus. In Chair), N. C. C., Choukri, K., Declerck, T., Dogan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).