Neural Machine Translation with Pluggable Denoise Function

Chen Xia
chenxia@cs.cmu.edu
Siyuan Wang
siyuanw@cs.cmu.edu

December 14, 2018

Abstract

In this project, we propose a method that allows us to use pretrained neural machine translation model, and by only adding a denoise function, the nmt model could normalize the text, and translate the text as the clean data. Also, we reduce the large number of parameters needed to re-train a denoise nmt model by fix all the parameters except for the denoise function itself. By reducing the parameters, we achieve the same BLEU score as naive domain adapation which fine-tunes the whole model on MTNT dataset.

1 Introduction

Non-standard text in neural machine tranlsaltion system will cause great troubles. There are some methods proposed to denoise manually by doing text normalization. [2] finetunes the model on MTNT datasets with the whole pretrained nmt model, and we reproduce the model baseline.

1.1 Datasets

The standard datasets used for pre-training NMT models consist of europarl-v7 and news-commentary-v10 corpora for the en/fr language pair, from which we created our train data and dev data after Moses corpus cleaning. There are 2,092,069 train samples and 1571 dev samples. We evaluate the model on the newstest2014 test set to compare with the results reported in [2]. The noisy text dataset we use is the MTNT dataset, which is built from noisy comments on Reddit1 and professionally sourced translations. We use the provided train split for direct finetuning and denoise layer training, and evaluate on the provided test split.
MTNT is a dataset based on reddit comment, and [2] select the parallel corpora and monoligual corpora.

2 Model Description

2.1 Baseline Model

All our models are implemented with the faiseq toolkit [1] by Facebook Research [1]. We train several standard NMT models on the standard datasets mentioned above to get the baseline results.

First, we run a simple seq2seq NMT model, largely based on the model configuration in the MTNT paper [2]. The encoder is a bidirectional LSTM with 2 layers, and the decoder is a 2 layered LSTM. However, we use simply dot-product attention. The encoder embedding dimension is 512, all other

[1]https://github.com/pytorch/fairseq

dimensions are 1024. We tie the target word embeddings and the output projection weights. We train with Adam optimizer, learning rate 0.001 and dropout probability 0.3. We use subword-nmt toolkit [2] for BPE, with each side 16,000 BPE size following the same setting as MTNT paper.

The second baseline model we run is a convolutional sequence to sequence model (FConv), as described in [1]. The configuration details we use can be found here [3]. For this model, we use a combined BPE model on both source and target languages with size 40,000, as recommended by the paper.

2.2 Naive Adaptation

With an NMT model trained on the standard datasets, we do naive domain adaptation by simply finetuning with the MTNT train data. We train with Adam optimizer but with 0.0001 learning rate until convergence.

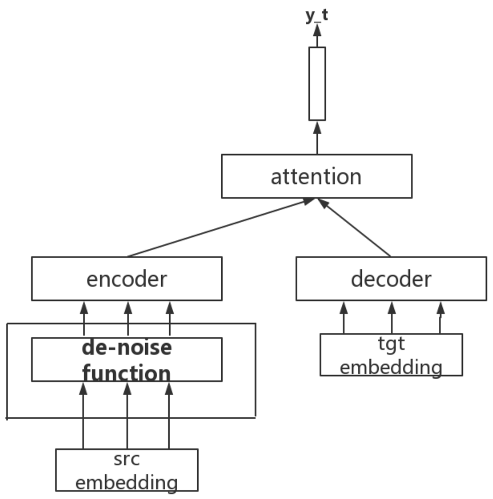

Figure 1: model architecture

2.3 Denoise Function

With a pre-trained seq2seq model, we fix all the parameters and add a denoise layer after the embedding layer in the encoder. We then only train the denoise layer with MTNT train data.

Our hypothesis is that this denoise function can learn how to transform a noisy source sentence to a clean one, so that the NMT model trained on clean corpora can have better and more robust performance.

To find a denoise function that fits in our settings, it should have the following properties: (1)handle variable length (2)have kind of encoding capability to normalize the input noisy text embedding. We choose a **bidirectional lstm with 1 layer as our denoise function**. Input size is the same as the embedding size, and the hidden size is half of the input embedding size, so it can output the same size of vector in our pretrained nmt model. We also experiment with adding a residual layer by adding the input to the output of the denoise layer, which we hope can help restrain the denoise layer output as word embeddings.

[2]https://github.com/rsennrich/subword-nmt

[3]https://github.com/pytorch/fairseq/blob/9dd8724577fd01511a59df962e856a7e30be4618/fairseq/models/fconv.py#L737

	Model	en-fr	fr-en
newstest2014	BiLSTM	34.8	29.8
	FConv	33.9	29.8
	BiLSTM(std) + finetune(mtnt)	34.3 (1ep) 33.6 (10ep) 33.3 (20ep)	29.8 (1ep) 29.5 (10ep) 29.3 (20ep)
MTNT	BiLSTM	24.6	23.9
	FConv	23.4	24.4
	BiLSTM(std) + finetune(mtnt)	29.5 (1ep) 32.6 (10ep) 33.6 (20ep)	31.2 (1ep) 34.6 (10ep) 35.4 (20ep)
	BiLSTM(std) + denoise(mtnt)	29.0	27.5
	BiLSTM(std) + res-denoise(mtnt)	29.4	27.4

Table 1: BLEU scores for baseline and finetuned models on newstest2014 and MTNT test

3 Experiment Results

Baseline and Naive Adaptation As shown in Table 1, BiLSTM and FConv trained on the standard dataset (coded "std" in the table) are comparable in performance on newstest2014 and MTNT test. As BiLSTM has fewer parameters than the FConv model and takes less time to train on a single GPU, we decided to use the BiLSTM as baseline for other experiments.

Our experiments show that naive finetuning the whole model with MTNT train data performs quite well. The BLEU score can be greatly improved with more epochs than [2] (which only finetune one epoch using SGD).

Denoise Function As shown in Table 1, denoise function achieves roughly the same BLEU score as finetune 1 epoch (reproduced by us, and reported by [2] as well). However, our model only needs to train the parameters of the denoise layer. In this regards, the denoise function can achieve decent improvement without fine-tuning the other parameters. Despite having comparable BLEU score as finetune-1epoch and lower than finetune more epochs, we think this is still a meaningful result because we keep the pre-trained NMT model parameters fixed, so that it can maintian the same petfrance on the original task. As shown in Table 1, with more epochs of fine-tuning, the model performance on the standard corpus newstest2014 keeps decreasing. By having our denoise function as a "pluggable" layer in the NMT model, we can greatly reduced model parameter size for multitasking scenarios.

In the case where we have multiple different noisy copora to adapt to, we can simply have one pre-trained standard NMT model, and train only a denoise layer for each corpus, which is more efficient and still achiieve reasonable performance. This project is a promising start for applications along this line.

4 Discussion and Analysis

4.1 What improved with finetuning and denoising?

Translation result comparison We pick up the sentence that is generated by the following model:**(a)**pretrain on standard datasets, finetune on MTNT datasets **(b)**pretrain on standard datasets, then only train denoise function on MTNT dataset **(c)**cotrain nmt + denoise function on standard datasets, then finetune only denoise function. From [Table 2], among three models, model(c) generates very interesting results, which turns "stressed" into "very sorry", which is more precise. Both finetune and denoise can improve the translation quality for the adjective "stressed" and time particle "up until now". More interestingly, the noisy text in French 'jusqu'a maintenant' should be 'jusqu'à maintenant', and our model tranlates it into 'up to now', which is quite close to 'up until now'.

source	mais bon la je re stresse car j’ai aucune idée de comment contribuer car j’ai fait quasi que de la théorie jusqu’a maintenant
target	But anyway I’m stressed again because I have no idea how to contribute since I mostly did theory up until now.
baseline	But I am **happy** because I have no idea how to contribute, because I have made almost the same theory **as before**.
baseline + finetune 20 epochs	But I’m **tired** because I have no idea how to contribute because I did almost like theory **until now <no period>**
baseline + residual_denoise	But **I’m sorry** because I have no idea how to contribute because I have made almost the theory **until now**.
cotrain + residual_denoise	However, **I am very sorry**, because I have no idea how to contribute, because I have almost had theory **up to now**.

Table 2: Naive domain adaptation vs denoise function

reference target	"I’m": 50, "it’s": 96, "I don’t": 31, "doesn’t": 20, ’I do not’: 1
baseline	’I do not’: 33, ’it is not’: 19, ’not have’: 11, ’That is’: 9
baseline + finetune 20 epochs	"I’m": 63, "it’s": 134, ”: 19, "I don’t": 41, "doesn’t": 17 ’I do not’: 2, ’not have’: 1
baseline + resdiual_denoise	’I do not’: 30, "it’s": 3, "I’m": 30, ’That is’: 6, "I don’t": 12 ’it is not’: 19, ’not have’: 8

Table 3: n-grams statistics under/over-generated

Frequency comparison of example n-grams To understand what actually changed from baseline to the generated translations after finetune and denoise, we count the number of occurances of some representative n-grams (contracted form and expanded form) in the generated translations. As shown in Table 3, there are many contracted forms such as "I’m", "doesn’t" in the reference target sentence, which is supposed to be noisy/informal. The baseline model (trained only on standard datasets) clearly under-generates these n-grams, but over-generates many expanded ones such as "I do not", "it is not". As for the model finetuned on MTNT train data, occurances of the contracted n-grams are similar to the reference target, and over-generated. For our model (baseline + residual_denoise), the statistics are more balanced, fewer expanded n-grams than the baseline, and slightly fewer contracted ones than the finetuned model, which is good (not over-fitted).

4.2 Denoising Effects

Top K scores with cosine similarity

To evaluate whether the denoise function is really doing the job, we perform the following analysis: we pick out the trained denoise function on MTNT data, and then use this part for the analysis. First, we find some sentence with noisy text, such as *’sutecthat’*, which should be *"such that"*. Then we put this sentence into the denoise function, and get the hidden vector of each time step. We then compare the **cosine similarity** of each subword in the sentence with the whole vocabulary (of subwords) embedding. Finally, we find top k similar subwords. Then we pick the top1 and top2

source	noisy word	Top 1 word	Top 2 word
I youre still gold while onetricking a champ it means your macro isn’t good.	champ	champion	champions
😂 is this in reference to the hair?	😂	&quot;	&apos;
I go to Meetup groups at a Panera and I love the broccoli cheese soup but not the 800mgs of sodium	800mgs	800m grams	8Mgs

Table 4: Top k similar word embeddings using cosine similarity

	standard dataset	mtnt (direct run on test)	mtnt (train with denoise)
denoise + residual + nmt	28.6	23.9	29.1
cotrain denoise + residual + nmt	29.8	24.17	29.6

Table 5: experiment on whether denoise function is really "do the job", instead of more parameters

similar subwrods respectively and concatenate the subwords to form top1 and top2 words (which is sequential, and easy to concatenate). From the result, we could see that **"champ"** could be converted from **"cham@@"** to **"champion"**, which is quite good.

Denoising or just more parameters To investigate how much of the improvement is contributed by simply additional parameters and how much by the denoise function itself, we run a control experiment where we add the denoise layer from the beginning when pretraining with the stardard datasets, and then fine-tune with MTNT train data with all parameters fixed except the denoise layer. As shown in Table 6, due to one additional layer in the encoder, the pretraining results are indeed slightly better on both newstest2014 and MTNT. Aftet fine-tuning the denoise layer with MTNT train data, the BLEU is still better than our model, which means the denoise layer is still doing its job and contritbues the same amount of improvement as before.

5 Conclusion

We proposed a method to utilize the existing neural machine translation model, and by adding denoise function, to translate noisy text into another language. The denoise function has the ability to normalize data as well as reduce the parameter in all to train a robust model. Hence, based on this framework, we could design more complicated and robust denoise functions to adjust to different set of noisy texts.

Bilstm denoise function is just an easy function to utilize. In the future, in order to denoise the noisy text, the denoise function could be explored in more advanced way(e.g. autoencoder, adverserial training). And the key point is to make the whole model robust enough to encouter various noisy text.

Acknowledgements

Thanks **Paul Michel** for all the guidance.

References

[1] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, *Convolutional Sequence to Sequence Learning*, in Proc. of ICML, 2017.

[2] P. Michel and G. Neubig, *Mtnt: A testbed for machine translation of noisy text*, arXiv preprint arXiv:1809.00388, (2018).