Multi-Domain Neural Machine Translation Implementation

Phan Anh Tu
Matrikel-Nr: 3698514
anhtu@cl.uni-heidelberg.de

Abstract

A novel multi-domain neural machine translation (NMT) model which apply word-level adaptive and layer-wise domain mixing to modify original transformer architecture and using individual modules for each domain is proposed in (Jiang et al., 2019) to overcome the lacks adaptation to individual domain when enforcing one encoder to learn shared embedding across domain. This project will try to implement the proposed model to achieved report result on original paper.

1 Introduction

Recently, the multi-domain neural machine translation (NMT) models are proposed to address the annotated parallel sentences limitation in certain domain when training. But the result of these models are not well exploit the domain-specific knowledge of each individual domain. There are two approach to address this drawback. The first approach is to assign domain related weights to different samples during training. The second approach is to design specific encoder-decoder architecture for NMT models. The proposed model use individual modules for each domain to capture domain-specific knowledge which addresses the limitation on the second approach when using one single encoder to learn shared embedding. Meanwhile, the first approach can be used as complementary to proposed model. By modifying multi-head dot-product attention modules in original transformer architecture for different domains, the domain knowledge sharing as well as the capturing fine-grained domain-specific knowledge are achieved effectively using the proposed model. Their experiments report show that in several NMT tasks, the proposed model outperforms existing models. Meanwhile, there is lack of the implementation to experience with with proposed model in original paper. The aim of this project will try to implement the proposed model as well as the do some experiments to get results as reported in original paper. The main contribute of this project are as follow:

- Prepare dataset of English-to-German tasks as well as pre-process data (tokenizer, encoded sentence using byte-pair encoding)
- Modifying transformer implementation to install proposed model
- Do some experiments with implemented domain-mixing model to compare with original transformer

The remainder of this report is organized as follows. Section 2 will briefly introduce the proposed model, the main contribution of proposed model in modifying transformer architecture. Some implementation notes is described in Section 3. Section 4 discuss the experiment results when comparing with the result of origin paper. Finally, Section 5 concludes the report and describe the future works.

2 Proposed model

The main contribute of proposed model is modifying the linear transformation in Multi-head self attention module in transformer architecture. In detail, instead of just using word-embedding as input, they plug domain proportion to each word to evaluate the domain of each word. Instead of using one weight projections matrices to calculate three matrices query, key and value, each domain have its own matrix and the result of linear transformation is then weighted sum with domain proportion to get final result. Figure 1 shows the example of modification to calculate linear transformation of matrix query (Q) with 3 domain; the input sentence has two words. $\mathcal{R}^T$ is parameter to get domain proportion matrix $\mathcal{D}$ of each word $\mathcal{D} = (1-\epsilon)\cdot softmax(\mathcal{R}X)+\epsilon/k$ with $\epsilon \in (0,1)$. Each column of matrix D is proportion of each word in this domain. And matrix Q^1 is a projection of two word in domain one, similarly, Q^2 is a

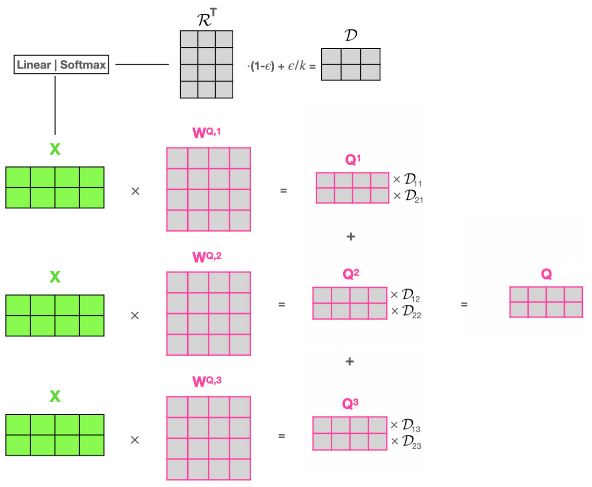

Figure 1: Matrix view of Word-Level Adaptive Domain Mixing with 3 domains

projection of two word in domain two and Q^3 is a projection of two word in domain three. These projection matrices is obtained by multiply the input word embedding with three different parameter matrices ($W^{Q,1}$, $W^{Q,2}$, $W^{Q,3}$) which are corresponding to three domain. Because the first row is the projection of first word so we will multiply with domain proportion of first word which is first element of column 1 in matrix D. Applying same technique, the projection matrix of 3 domains after multiplying with domain proportion is averaged to get projection matrix Q.

For training proposed model, beside the loss compute as original transformer architecture L_{gen} which is the cross entropy loss over training data $\{x_i, y_i\}_{i=1}^n$ between output sentence and target sentence, L_{mix} loss is the cross entropy loss over words and domain labels. The domain label of a sentence is the domain that this sentence belong to in dataset. All words in a sentence have domain same as this sentence. Specifically, each word in sentence x_i has a domain label J ($J-th$ domain); given embedding x_{ik} of a word k in sentence x_i, cross entropy loss of its is: $-\log(\mathcal{D}_J(x_{ik}))$; L_{mix} is the sum of the cross entropy loss over all such pair of word and domain label in sentence x_i. As a result, the loss for training the proposed model is $L^* = L_{gen} + L_{mix}$. Therefore, domain proportion layers $\mathcal{D}$ is trained solely and not intervene in the training of the translation model. In other word, matrix $\mathcal{R}^T$ is updated by L_{mix}, meanwhile L_{gen} is used to update the parameters of translation model as shown in Figure 2.

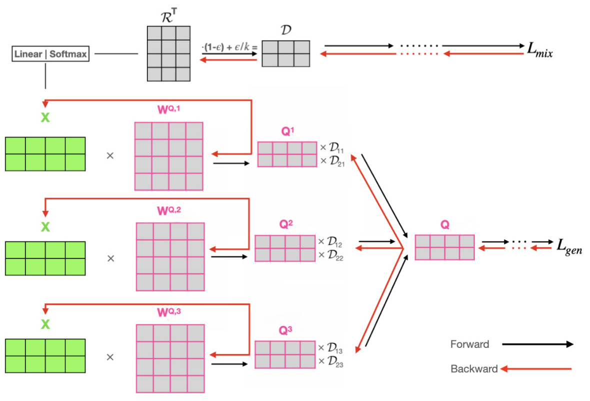

Figure 2: Computational graph for training the domain proportion layers

3 Implementation

This project is implemented by *pytorch*. The original transformer architecture is implemented follow by the instruction in Attention is All You Need notebook (Trevett). The implementation of byte-pair encoding is taken from (Huang) and is modified to tokenize using MOSES script instead of tokenize by splitting by space.

The domain-mixing model is implemented base on the original transformer implementation. Specifically, Multi-Head-Attention-Layer and Position-Wise-Feedforward-Layer are modified. In Multi-Head-Attention-Layer, query, key, value input have their own $\mathcal{R}^T$ matrix to calculate the domain proportion matrix $\mathcal{D}$ and the matrix $\mathcal{D}$ of query is output to compute the L_{mix} loss. The domain proportion matrix $\mathcal{D}$ of encoder layer is average of domain proportion matrix $\mathcal{D}$ of Multi-Head-Attention-Layer and domain proportion matrix $\mathcal{D}$ of Position-Wise-Feedforward-Layer. Meanwhile, the domain proportion matrix $\mathcal{D}$ of decoder layer is average of domain proportion matrix $\mathcal{D}$ of Multi-Head-Attention-Layer and domain proportion matrix $\mathcal{D}$ of Multi-Head-Cross-Attention-Layer and domain proportion matrix $\mathcal{D}$ of Position-Wise-Feedforward-Layer. The domain proportion matrix $\mathcal{D}$ of encoder (decoder) module is the average of domain proportion matrix $\mathcal{D}$ of all encoder (decoder) layer. In architecture with domain proportion is plugged just into encoder, L_{mix} is computed by domain proportion matrix $\mathcal{D}$ of encoder module. In architecture with domain proportion is plugged into encoder and decoder, L_{mix} is computed by domain proportion matrix $\mathcal{D}$ of decoder module.

To take advantage of matrix multiplication in pytorch. To compute projection matrix of query, key and value Q, K, V, when multiplying the projection matrix with domain proportion, the matrix multiplication is implemented instead of using `for` loop.

The follow example show the implementation of `for` loop to calculate the projection matrix when multiplying with domain proportion in domain 1.

```
for i_q_b in range(q_1.shape[0]):
    for i_q in range(q_1.shape[1]):
        q_1[i_q_b, i_q, :]
            *= d[i_q_b, i_q, 0]
```

Suppose we have projection matrix `q_1` with shape `(batch_size, sentence_length, hid_dim)`. As describe in Section 2, the i_q^{th} row of projection matrix `q_1` in each batch `q_1[i_q_b, i_q, :]` need to be multiplied by the domain proportion of i_q^{th} row of domain 1^{st} (`d[i_q_b, i_q, 0]`).

The following is the implementation example when applying pytorch matrix multiplication to compute the same result as `for` loop implementation below. We will repeat the first column of matrix $\mathcal{D}$ which is the domain proportion in domain 1 to obtain the matrix `i_dq` with the same shape as `q_1` and then apply matrix multiplication between `i_dq` and `q_1`. Because `i_dq` and `q_1` has the same size, pytorch will multiply each element of `i_dq` with corresponding element of `q_1`

```
i_dq = torch.repeat_interleave(
                    d[:, :, i_d],
                    q_1.shape[2],
                    dim=1
            ).view(q_1.shape)
q_1 *= i_dq
```

With small modification in implementation, the training time is increase significantly. When testing with same resource (using cpu), the forward time when implement with `for` loop is 29.29s compare to 12.45s when implement with matrix multiplication. The backward gradient time when implement with `for` loop is 486.66s compare to 24.24s when implement with matrix multiplication.

4 Experience result

This project experience with English-to-German task that the dataset has two domain: News and TED where News domain data is collected from Europarl (Koehn, 2005) and TED domain data is collected from IWLST (Cettolo et al., 2014). The Table 1 shows the number of sentence in training, validation, and testing sets of each domain in dataset.

Domain	Train	Valid	Test
News	184K	18K	19K
TED	160K	7K	7K

Table 1: The number of sentences in the dataset

This project will experience with four models. There are:

- **direct-training**: Transformer model trained using merged training data from two domains
- **E/DC**: The modified transformer which the domain proportion is plugged in both encoder and decoder module
- **E/DC-with-int**: same as **E/DC** but apply *xavier_uniform* to initialize parameters
- **Encoder**: The modified transformer architecture which the domain proportion is just plugged in encoder module

The experience is done on server with Nvidia GTX 1080 Ti (Pascal, 11GB memory). Table 2 shows the time per step (training and validating) when training four models. Figures 3 and 5 show the loss on training and validation set of four models (the loss when training with direct-training is jump up at the middle because the batch size of these steps increase from 32 to 64); the accuracy is shown on Figures 4 and 6. We can see training with domain proportion model seem to performs stable on validation set (In Figures 5 and 6 the line of direct-training usually jump up and down). With the suitable parameter initialization method, the model can get better performance (the loss on both training and validation set of E/DC-with-int is smaller than E/DC (the accuracy is higher)). Table 3 shows the BLEU score of four models.

The result is not good as original paper. For direct-training maybe because the initialize of parameters. The fail of E/DC model maybe because the L_{mix} loss is just calculated by domain proportion matrix of decoder module. Therefore there is just $\mathcal{R}^T$ parameters on decoder module is updated when training. The improvement of L_{mix} which is calculated by both domain proportion of encoder and decoder module is considered as future word. The result of when training encoder models is not good as original paper as well as not higher than direct-training as expected maybe because the parameters initialization method and the training time

Model	**Time per step**
direct-training	1959.03s
E/DC	9694.38s
Encoder	7454.45s

Table 2: Training time

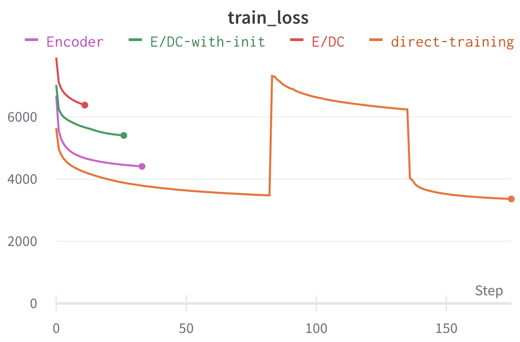

Figure 3: Training loss

is short (we can see that the *train_loss* as well as the *val_loss* still can decrease).

5 Conclusion

This project tried to modified original transformer architecture to implement word-level adaptive and layer-wise domain mixing and experience on training neural machine translation model to translate from English to German on two domain NEWS and TED. The improvement of implementation when plugging domain proportion to both encoder and decoder as well as experiencing with different parameters initialization methods to get result as original paper is consider as future works. In addition, the other tasks like translate from English to French, from Chinese to English as well as experience with other domain aware embedding based method

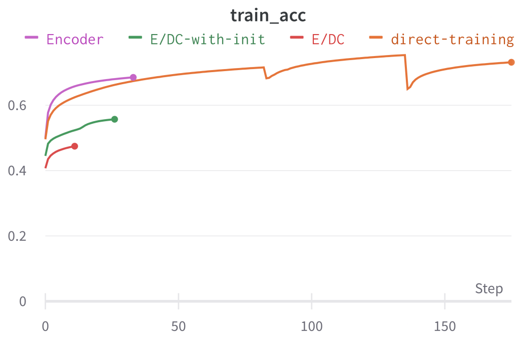

Figure 4: Training accuracy

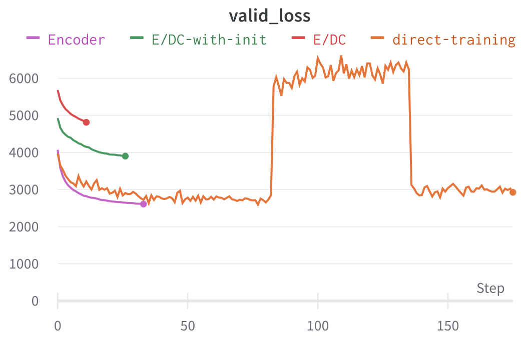

Figure 5: Validation loss

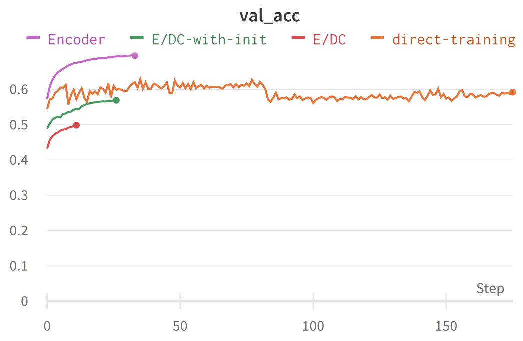

Figure 6: Validation accuracy

Method	News	TED
direct-training	19.0 (26.06)	25.0 (28.11)
E/DC	1.0 (27.58)	1.0 (30.33)
E/DC-with-init	1.0	1.0
Encoder	15.0 (27.78)	25.0 (30.30)

Table 3: BLEU score in English-to-German. The number in bracket bracket is the result in original paper.

such as Multitask Learning, Adversarial Learning, Partial Adversarial Learning, Word-Level Domain Context Discrimination is also considered to complete get all the experience in original paper.

References

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign*, pages 2–17, Lake Tahoe, California.

Yu-Hsiang Huang. Attention is all you need repository.

Haoming Jiang, Chen Liang, Chong Wang, and Tuo Zhao. 2019. Multi-domain neural machine translation with word-level adaptive layer-wise domain mixing.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand.

Ben Trevett. Attention is all you need notebook.