Adversarial Neural Machine Translation

Yirui Wang Xuan Zhang Channing Kimble-Brown

Computer Science
Whiting School of Engineering
Johns Hopkins University
{*yrwang, xuanzhang, ckimble3*}*@jhu.edu*

Abstract

Different from general Neural Machine Translation(NMT) networks, which aim to maximize the likelihood of the human translation, Generative Adversarial Networks(GANs) minimize the distinction between human translation and the translation generated from an NMT model. The GAN introduces an adversary to discriminate the real and fake translation, and it will be trained together with the generator. In this project, we implemented the GAN[1], with an attention-based Recurrent Neural Network(RNN) as the generator, and a 2-layer Convolutional Neural Network(CNN) as the discriminator. By comparing the performance of traditional NMT model, we conclude that the Adversarial-NMT can achieve better translation.

1 Introducation

Neural Machine Translation (NMT) has become more and more popular in both academic and industry. Despite its success, the translation quality of NMT system is still unsatisfied and there remains a large room for improvement. The NMT model aims to maximize the probability of the target ground-truth sentence given the source sentence. Such an object does not guarantee the translation results to be natural and sufficient like human-translations. Some previous works tried to alleviate this limitation by reducing the objective inconsistency between NMT training and inference. Thus, they directly maximizing BLEU (Papineni et al., 2002). Some improvement is observed, but the objective still cannot bridge the gap between NMT translations and human-generated translations. Thanks to the success of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), several works have been done to adopt GANs for NMT. The latest two works are (Lijun Wu et al., 2017) and (Zhen Yang et al., 2017).

In this project, we manage to investigate how Adversarial-NMT system works and implement an Adversarial-NMT model according to the work of (Lijun Wu et al., 2017). And finally compare our Adversarial-NMT with traditional NMT models. Due to limited computation resource and time, especially the GPU memories, we can only run our model on a small German to English dataset for some epochs, which consisting about 10K training sentence pairs and 3K validation pairs.

2 Generative Adversarial Network

The overall framework of our Adversarial-NMT is shown in Figure 1. Let x and y be a binlingual aligned sentence pair for training, where x_i is the i-th word in the source sentence and y_j is the j-th word of the target sentence. Let y' denote the translation sentence out from an NMT system for the source sentence x. The goal of Adversarial-NMT is to force y' to be as similar as the human translation y. While the

[1]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT

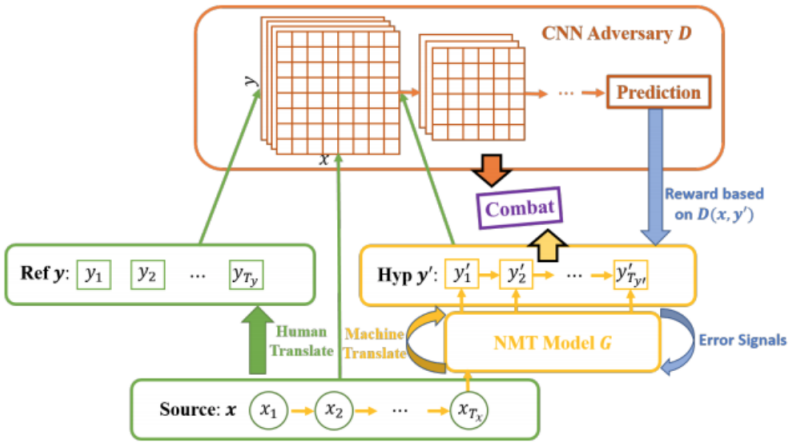
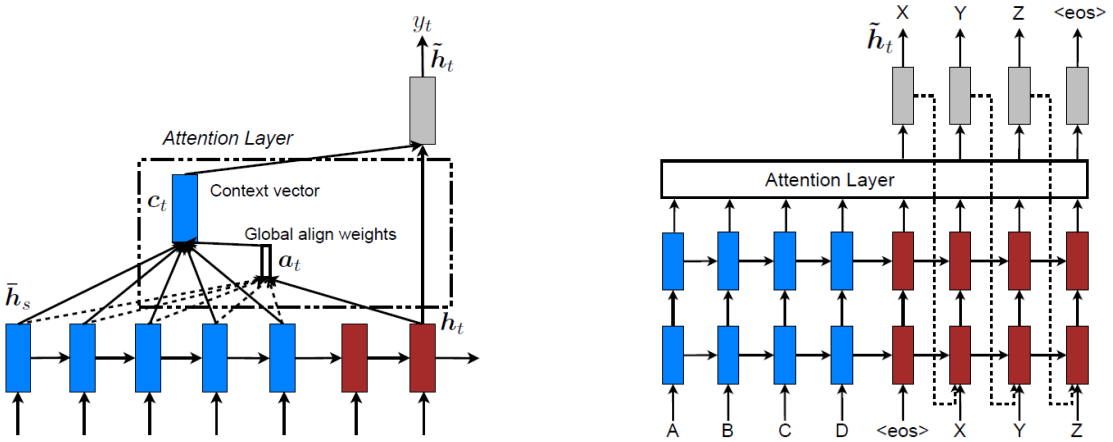


Figure 1: The Adversaria-NMT framework [1]



(a) Global attentional model [4]

(b) Input-feeding approach [4]

Figure 2: The NMT Model

goal of the adversary is to differentiate human translation from machine translation, and the generator tries to produce a target sentence as similar as human translation so as to fool the adversary.

2.1 Generator

We adopt the Recurrent Neural Network (RNN) based encoder-decoder[2] as the generator (NMT) model to seek a target language translation y_0 given source sentence x. In particular, a probabilistic mapping $G(y|x)$ is firstly learnt and the translation result $y' \sim G(\bullet|x)$ is sampled from it. An **attentional mechanism** has been introduced to improve neural machine translation (NMT) by selectively focusing on parts of the source sentence during translation (Luong et al., 2015). There are two kinds of attentional mechanism: a global approach which always consider all source words, while a local one only looks at a subset of source words at a time. In our project, a gloabl attentional approach was used to improve our generator (NMT) model. According to (Luong et al., 2015), the computing path is going from $h_t \rightarrow a_t \rightarrow c_t \rightarrow \widetilde{h}_t$. So, firstly, the RNN hidden unit h_t can be abstractly computed as:

$$\boldsymbol{h_t} = f(\boldsymbol{h_{t-1}}, \boldsymbol{s})$$

[2]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT

where f computes the current hidden state given the previous hidden state (in our project this f is a **LSTM** unit). And s is a set of source side hidden states, which are consulted throughout the whole translation process by using attentional mechanism.

As mentioned before, the idea of global attentional model is to consider all the hidden states of the encoder when deriving the context vector c_t. In this model type, an alignment vector a_t is derived by comparing the current hidden state h_t with all the source hidden state $\widetilde{h}_s$:

$$\begin{aligned} \boldsymbol{a}_t(s) &= align(\boldsymbol{h_t}, \widetilde{\boldsymbol{h}}_{\boldsymbol{s}}) \\ &= \frac{exp(score(\boldsymbol{h_t}, \widetilde{\boldsymbol{h}}_{\boldsymbol{s}}))}{\sum_{s'} exp(score(\boldsymbol{h_t}, \widetilde{\boldsymbol{h}}_{\boldsymbol{s'}}))} \end{aligned}$$

Here, score is a content-based function, which provides three different choices:

$$score(\boldsymbol{h_t}, \widetilde{\boldsymbol{h}}_{\boldsymbol{s}}) = \begin{cases} \boldsymbol{h_t^T}\hat{\boldsymbol{h}}_{\boldsymbol{s}} & \text{dot} \\ \boldsymbol{h_t^T}\boldsymbol{W_a}\hat{\boldsymbol{h}}_{\boldsymbol{s}} & \text{general} \\ v^T tanh(\boldsymbol{W_a}[\boldsymbol{h_t}; \hat{\boldsymbol{h}}_{\boldsymbol{s}}]) & \text{concat} \end{cases}$$

In our project, we choose **general** form score for further computation. The next step is to compute the context vector c_t given the alignment vector as weights. Thus, the context is computed as the weighted average over all the source hidden states.

The last step is to compute the attentional vector $\widetilde{h}_t$. Given the target hidden state h_t and the source side context vector c_t, we can simply concat these two vectors and then feed into a linear layer to produce our final attentional vector $\widetilde{h}_t$:

$$\widetilde{h}_t = tanh(\boldsymbol{W_c}[\boldsymbol{c_t}; \boldsymbol{h_t}])$$

Now, the attentional vector can be fed through the softmax layer to produce the predictive distribution:

$$p(y_t|y_{<t}, x) = softmax(\boldsymbol{W_s}\widetilde{\boldsymbol{h}}_{\boldsymbol{t}})$$

Furthermore, according to (Luong at el., 2015), in the global attentional approach, the attentional decisions are made independently, which is considered as suboptimal. This, they introduce an input-feeding approach in which attentional vectors $\widetilde{h}_t$ are concatenated with inputs at the next time step (as shown in Figure 2b).

2.2 Discriminator

In order to measure the translative matching degree of the source-target sentence pair, we turn to CNN[3] for this task, since with its layer-by-layer convolution and pooling strategies, CNN is able to accurately capture the hierarchical correspondence of source-target sentence pairs at different abstraction levels.

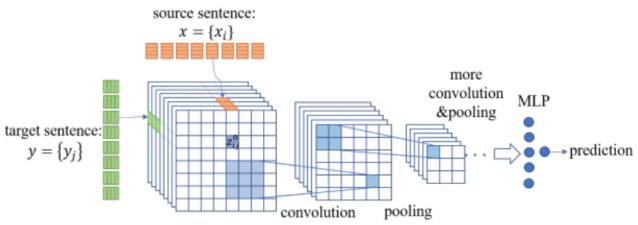


Figure 3: The CNN adversary framework [1]

The general structure is shown in Figure 3. We first concatenate the embedding vectors in the source sentence x and the target sentence y, and each word y_j in the target sentence y will be concatenated to every word in x. This constructs a 2D

[3]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT

image-like representation with the "image" size of the square of the sequence length, and there are 2 x the embedding size channels in total for one sample input to the CNN.

Then we perform convolution on a 3x3 window, with the purpose of capturing the correspondence between segments in source and target sentences pushing the result to the sigmoid activation function. After that we perform a max-pooling in nonoverlapping 2x2 windows. We go on for two layers of convolutions and max-pooling, aiming at capturing the correspondence at different levels of abstraction, but also avoiding to make the network to complex. The extracted features are then fed into a multi-layer perceptron (MLP) and another fully connected layer to output a vector with only two elements corresponding to the two classes(human-translated sentence and generated sentence) respectively, with sigmoid activation at the last layer to give the probability that the sentence pair is from ground-truth data. The optimization target of such a CNN adversary is to minimize the cross entropy loss for binary classification, with ground-truth data as positive instances while sampled data as negative instances.

2.3 Training

We use policy gradient algorithm to train Adversarial-NMT[4], to tackle the problem that the discretely sampled y' from the NMT model G makes it difficult to directly back-propagate the error signals from the adversary model D to G. For the convenience of explanation, let $p_{data}(x)$ represent the distribution of true translation data, let p_g represent the generator's distribution over data, while $p_z(z)$ is the distribution of noise z, which is the randomly initialized word embeddings. Let $G(z)$ be the output of the generator, and $D(z)$ is the probability that x is the ground-truth translation rather than data produced by the generator.

For the discriminator D, it is trained to better classify real translation from the fake translation from the generator G. So it should tend to output 1 for true translation and 0 for fake translation, as its output is the probability that the sentence is human-translated. That is,

$$\begin{cases} \text{if } x \sim p_{data}(x),\ D(x) = 1,\ E_{x\sim p_{data}(x)} \log(D(x)) \text{ is the maximum} \\ \text{if } x \sim p_z(z),\ D(G(z)) = 0,\ E_{x\sim p_z(z)} \log(1 - D(G(z))) \text{ is the maximum} \end{cases}$$

Define the value function

$$V(G, D) = \log(D(x)) + \log(1 - D(G(z))).$$

The best discriminator is

$$D_G^* = \arg\max_D V(G, D).$$

While the goal of the generator G is to fool the discriminator D that the sentence it generated is true translation. So the best generator should be

$$G_D^* = \arg\min_G(\arg\max_D V(G, D)).$$

It can be proven the best generator exists.

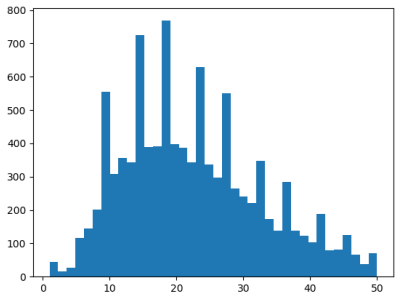
Based on above induction, we introduce the following loss for the training of the two models. We use cross entropy as the criterion. G aims to let D believe that the generated translation is ture data, thus the loss should be

$$\mathcal{L}_G = \mathbb{H}(1, D(G(z))).$$

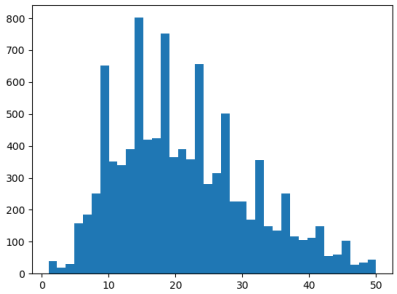
And the loss for training the discriminator should be

$$\mathcal{L}_D = \mathbb{H}(1, D(x)) + \mathbb{H}(0, D(G(z))).$$

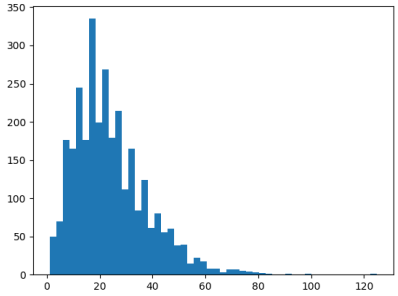
[4]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT



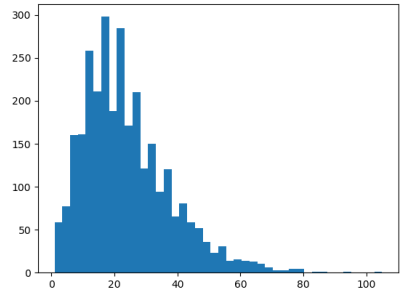
(a) English training sentence length distribution



(b) German training sentence length distribution



(c) English validation sentence length distribution



(d) German validation sentence length distribution

Figure 4: Sequence Length Distribution on Dataset

The above equations imply that the more likely y' to successfully fool D, i.e. larger $\mathcal{L}_D$, the larger reward the NMT model will get, and the fake training data (x, y') will correspondingly be more favored to improve the G.

3 Experiments

3.1 Data

We use the same German→English translation dataset as OpenNMT[5][6]. This dataset contains 12K sentence pairs in total, and we use 10K of them for training and the rest for validation. As we need to make sure the input sentences of the discriminator CNN are in the same length, we need to do padding for shorter sentences and discard some of the long sentences. As Figure 4 shows, the length of most English and German sentences is smaller than 35, and we remove the translation pairs where the English or German sentence is longer than 35, which constitue around 10% of the entire dataset.

3.2 NMT model

Due to computation resource limit, our traditional Neural Machine Translation (NMT) model is trained on the OpenNMT demo dataset, which consists 10K training sentence pairs and 2K validation pairs. Since the Adversarial-NMT would train two models (generator and discriminator) at the same time, the GPU requirement is relatively high. In order to check whether our model works, we have to make a trade-off between the translation quality and the computation cost. Thus, some settings are different from the instructions in HW5.

[5]https://github.com/OpenNMT/OpenNMT-py
[6]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT

3.2.1 Encoder

We reduce the embedding size to 50 to fit the limited GPU memory. Then, a one layer bi-directional LSTM (with a hidden size 1024) is used to encode the embedding sentences.

3.2.2 Decoder

Based on the description in (Luong et al., 2015) and HW5, we use a one layer LSTM model with attentional mechanism to decode the encoder output information. The same as the encoder, our decoder embedding size and hidden size are 50 and 1024, respectively.

3.2.3 Initialization and Optimization

We follow (Sutskever et at., 2014) and (Luong et al., 2015) in training the generator with similar settings: (a) our parameters are uniformly initialized in [-0.1, 0.1], (b) the normalize the gradient is rescaled whenever its norm exceeds 5.0. (c) we use a dropout probability 0.2 for our LSTMs as suggested by (Zaremba et al., 2015). The Adam optimizer is selected, with learning rate 0.001 as suggested in HW5. We use a small batch size 16 during the training and validation due to the limited GPU memory.

3.3 Adversary CNN

For the adversary D, the CNN consists of two convolution+batch normalization+relu+pooling layers. For the first convolution layer, we set the kernel size to 3x3 and the number of feature maps to 64, with stride 1 and padding size 1. The second convolution layer outputs 20 feature maps, using 3x3 convolutional window size, with stride 1 and padding size 1. The pooling window size for both layers is set to 2x2. Then we have two fully connected layers, first map 1280 features to 20 hidden nodes and then map them to 2 nodes. Finally we apply sigmoid function to the output of the CNN.

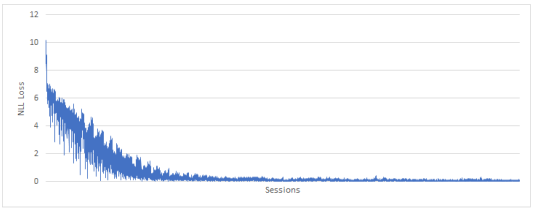
The input of the discriminator is a 3D tensor, with double-word-embedding-size channels, and the size of the representation on each channel is equal to the square of sequence length. We pad the sequences by zeros to increase their length to 35 as mention in the Data section. And the word embedding size should be the same as used in the NMT model, which is 50. Thus the input tensor should be in the size of 100x35x35.

3.4 Training

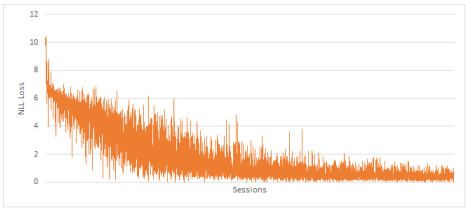
To train the GAN[7], we combine the generator and discriminator to train them together. The embedded source sequence batch first flows through the NMT model, and the generated fake translation and the ground-truth transaltion will be sent to the discriminator. As mentioned in section 2.3, we calculate different losses for G and D, and first update the parameters of the discriminator by backpropagation, then update the NMT model. We force 50% randomly chosen mini-batch data are trainied with Adversarial-NMT, while apply MLE principle(negative log-likelihood loss) to the other mini-batch data. The MLE acts as a regularizer to guarantee smooth model update, alleviating the negative effects brought by high gradient estimation variance introduced by the policy gradient algorithm [1].

We use batch size of 16, learning rate of 0.001 and Adam optimizer to train our model. And for comparison, we apply the same setting to train the NMT model alone.

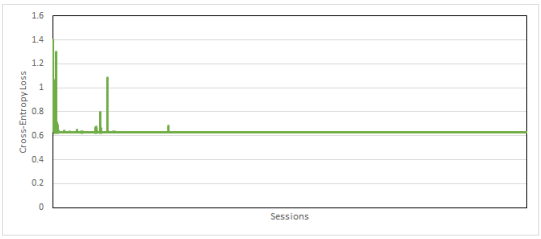
[7]Code Link: https://github.com/wangyirui/Machine-Translation/tree/master/NMT



(a) The training loss of simple G on 50 epochs



(b) The training loss of GAN's G on 50 epochs



(c) The training loss of GAN's D on 50 epochs

Figure 5: Sequence Length Distribution on Dataset

3.5 Results

In order to show our Adversarial-NMT has better performance than orginal NMT models, we compare its performance with the trained-alone generator with same experiment settings as mentioned above. Figure 5 illustrates the loss changes on each training data batch for 50 epochs of both Adversarial-NMT model and the simple NMT model. As we can see, the loss of GAN's generator on training set decreases more slowly than the simple generator with wilder fluctuation. This is due to the participation of the discriminator. The wild waves are like strong combats between G and D, D wants to achieve larger $V(G, D)$(see section 2.3), while G aims to successfully fool D, that is to make $V(G, D)$ smaller. As for the loss of D, except several violent fluctuations, it is stable during the training. This may due to the limited "forge" ability of G, which makes D easily differentiate fake sentences produced by G with human translation. Maybe with more data and more training time, we'll get a better generator.

With the trained GAN, the loss on validation set is 4.23 lower than the NMT model trained alone.

**We aware that something may be wrong with the generator. It may overfit the training data after several epochs, as after several epochs, the training loss continues decreasing while the validation loss starts rising continually until the end. In Addition, the training loss of the discriminator also acts weird. It drops to a small loss at the very beginning after several batches, and then stays at a same value at the most of time just like it stops learning. As the description of the implementation detail of the Adversarial-NMT model is relatively limited [1], we chose many parameters and structures of our model from past experience, and we didn't have enough time to fine-tune those variables. Thus we can not guarantee that our model shows perfectly the superiority of the Adversarial-NMT model.*

4 Conclusion

Adversarial-NMT adds a discriminator as a new component to the original NMT model to help it generate more natrual, sufficient and accurate translation, and directly minimize the difference between machine and human translation. We investigated the idea and the implementation of GANs in detail, and came up with the Adversarial-NMT model designed by ourselves. Then we conducted some experiments on our model using the German→English translation dataset. And the results on validation set imply that Adversarial-NMT model can produce a better generator than the original NMT model.

5 Future Work

There are many improvements can be done, but due to time and computing resources limitation, we haven't implemented them. As the first step, the CNN adversary network D can be initially pre-trained using the sampled data sampled from the NMT model and the ground-truth translation. We can also pre-train the generator independently. Parameters can be further tuned in both generator and discriminator model. As the discriminator is easy to overfit, it can be reconstructed to a simpler network, with fewer layers or even without the convolution and pooling layers.

References

[1] Wu, Lijun, et al. *Adversarial Neural Machine Translation.* arXiv preprint arXiv:1704.06933 (2017).

[2] Yang, Zhen, et al. *Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets.* arXiv preprint arXiv:1703.04887 (2017).

[3] Goodfellow, Ian J., et al. *Generative Adversarial Networks.* arXiv preprint arXiv:1704.06933 (2017).

[4] Luong, Minh-Thang, et al. *Effective Approaches to Attention-based Neural Machine Translation* arXiv preprint arXiv:1508.04025 (2015).

[5] Williams, Ronald J, et al. *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.* Machine learning, pp 229-256 (1992).