

Revisiting Robust Neural Machine Translation: A Transformer Case Study

Peyman Passban^{1,*},[†] Puneeth S.M. Saladi^{1,2,*} Qun Liu¹

¹Huawei Noah’s Ark Lab

²Faculty of Mathematics, University of Waterloo

passban.peyman@gmail.com

{puneeth.saladi, qun.liu}@huawei.com

Abstract

Transformers (Vaswani et al., 2017) have brought a remarkable improvement in the performance of neural machine translation (NMT) systems but they could be surprisingly vulnerable to noise. In this work, we try to investigate how noise breaks Transformers and if there exist solutions to deal with such issues. There is a large body of work in the NMT literature on analyzing the behavior of conventional models for the problem of noise but Transformers are relatively understudied in this context. Motivated by this, we introduce a novel data-driven technique called Target Augmented Fine-tuning (TAFT) to incorporate noise during training. This idea is comparable to the well-known fine-tuning strategy. Moreover, we propose two other novel extensions to the original Transformer: Controlled Denoising (CD) and Dual-Channel Decoding (DCD), that modify the neural architecture as well as the training process to handle noise. One important characteristic of our techniques is that they only impact the training phase and do not impose any overhead at inference time. We evaluated our techniques to translate the English–German pair in both directions and observed that our models have a higher tolerance to noise. More specifically, they perform with no deterioration where up to 10% of *entire* test words are infected by noise.

1 Introduction

NMT is the task of transforming a source sequence into a new form in a particular target language using deep neural networks. Such networks commonly have an encoder-decoder architecture (Cho et al., 2014a,b; Sutskever et al., 2014), in which an encoder maps a given input sequence to an intermediate representation and a decoder then uses this representation to generate candidate translations. Both encoder and decoder are neural networks that

are trained jointly. Due to the sequential nature of the NMT task, early models usually relied on recurrent architectures (Yang et al., 2020), or benefited from the sliding feature of convolutional kernels to encode/decode variable-length sequences (Kalchbrenner et al., 2014; Gehring et al., 2017).

Recently, Transformers (Vaswani et al., 2017) have shown promising results for NMT and become the new standard in the field. They follow the same concept of encoding and decoding but in a relatively different fashion. A Transformer is fundamentally a feed-forward model with its unique neural components (self-attention, layer norm, etc) that altered the traditional translation pipeline accordingly. It is expected if such a new architecture would behave differently than its recurrent or convolutional counterparts, and our goal in this research is to study this aspect in the presence of noise.

NMT engines trained on clean samples provide high-quality results when tested on similarly clean texts, but they break easily if noise appears in the input (Michel and Neubig, 2018). They are not designed to handle noise by default and Transformers are no exception. Many previous works have focused on this issue and studied different architectures (Li et al., 2019). However, in this work, we particularly focus on Transformers¹ as they are relatively new and to some extent understudied.

A common approach to make NMT models immune to noise is *fine-tuning* (*FT*), where a noisy version of input tokens is intentionally introduced during training and the decoder is forced to generate correct translations despite deformed inputs. *FT* is quite useful for almost all situations but it needs to be run with an optimal setting to be effective. In our experiments, we propose a slightly different learning-rate scheduler to improve *FT*. We also define a new extension that not only modifies

*Equal contribution.

[†] Work done while Peyman Passban was at Huawei.

¹We assume that the reader is already familiar with the Transformer architecture.

input words but also adds complementary tokens to the target side. We refer to this extension as *Target Augmented Fine-Tuning (TAFT)*, which is the first contribution of this paper.

In our study, we realized that data augmentation techniques (*FT* and *TAFT*) might not be sufficient enough and we need a compatible training process and neural architecture to deal with noise. Therefore, we propose *Controlled Denoising (CD)* whereby noise is added to source sequences during training and the encoder is supposed to fix noisy words before feeding the decoder. This approach is implemented via an auxiliary loss function and is similar to adversarial training. *CD* is our second contribution.

CD only takes care of noise on the encoder side. We also propose a *Dual-Channel Decoding (DCD)* strategy to study what happens if the decoder is also informed about the input noise. *DCD* supports multi-tasking through a 2-channel decoder that samples target tokens and corrects noisy input words simultaneously. This form of fusing translation knowledge with noise-related information has led to interesting results in our experiments. *DCD* is the third and last contribution of this work.

The remainder of the paper is organised as follows: We first review previously reported solutions for the problem of noise in NMT in Section 2. Then, we present details of our methods and the intuition behind them in Section 3. To validate our methods, we report experimental results in Section 4. Finally, we conclude the paper and discuss possible future directions in Section 5.

2 Related Work

Fine-tuning (*FT*) is one of the most straightforward and reliable techniques to protect NMT systems from noise. Berard et al. (2019), Dabre and Sumita (2019), and Helcl et al. (2019) studied its impact and showed how it needs to be utilized to boost NMT quality.

Adversarial training is another common solution to build noise-robust models. Cheng et al. (2019) proposed a gradient-based method to construct adversarial examples for both source and target samples. Source-side inputs are supposed to attack the model while adversarial target inputs help defend the translation model. In their model, a candidate word is replaced with its semantically-close peer to introduce noise. This way, the neural engine visits different forms of the same sample, which extends

its generalization. In other words, the network is trained to deliver the same, consistent functionality even though it is fed with different forms of a sample. Although this strategy showed promising results, in our setting we replace input words with *real* noisy candidates instead of synonyms or semantically-related peers. We find this way of adding noise more realistic and closer to real-world scenarios.

Karpukhin et al. (2019) experimented another idea by generating adversarial examples using synthetic noise. Their proposed architecture relies on Transformers but the encoder is equipped with a character-based convolutional model (Kim et al., 2016). This work is one of the few attempts that studied Transformers' behaviour in the presence of noise. However, their results are based on relatively small datasets. We know that NMT models' performance could change proportionally with a change in the size of training sets. Therefore, we used larger datasets in our experiments.

The application of adversarial training is not limited to the aforementioned examples. Cheng et al. (2018) defined additional loss functions which force the encoder and decoder to ignore perturbations and generate clean outputs. This idea is similar to our *CD* approach, but the underlying architecture is different. Cheng et al. (2018) only reported results on recurrent NMT models.

Providing better representations is as important as designing tailored training strategies for noise-robust models. A group of researchers focused on how different segmentation schemes and encoding techniques can play a role. Sennrich et al. (2016) and Michel and Neubig (2018) showed that subwords are better alternatives than surface forms (words) to handle perturbations and out-of-vocabulary words. Belinkov and Bisk (2018) comprehensively studied this by using different character- and subword-based representations in different architectures. Sakaguchi et al. (2017) also carried out a similar investigation where they proposed a new encoding that is invariant to the order of characters.

Besides these approaches, translating noisy inputs can be viewed as a two-pass process performed via two connected neural networks. The first one acts as a monolingual translator to correct noisy inputs and the second one is an engine that consumes denoised sequences to generate clean translations (Sun and Jiang, 2019; Zhou et al., 2019). This

idea can be implemented as an end-to-end, differentiable solution or as a pipeline, but it should be noted that such a mechanism could be hard to deploy or slow(er) to run in practice.

3 Methodology

This section covers details of our proposed methods. *FT* is a well-known technique so we skip its details and only focus on *TAFT*, our own extension of it (Section 3.1). Besides *FT* and *TAFT* that leverage data, we introduce *CD* (Section 3.2) and *DCD* (Section 3.3), which modify the training procedure as well as the neural architecture of Transformers.

3.1 Fine-Tuning Transformers

FT simply exposes an already-trained translation engine to noise during training in the hope of extending its coverage at test time. This simple idea is quite effective, but it requires to be run with an optimal setting, e.g. the type/amount of noise added to the training set directly impacts performance. It is also crucial to find an optimal number of iterations. Overrunning *FT* could hurt quality and be as costly as training a new model, and running it for an insignificant number of iterations might not be enough to reveal its power. Clearly, a better choice of these hyper-parameters leads to better results, but in addition to this empirical side of *FT*, we realized that it can be boosted even more via a simple modification.

FT only alters source sequences. In our extension (*TAFT*), we change the target side as well by appending clean versions of perturbed source words to the target sequence. Table 1 provides an example for this form of data augmentation. An ordinary model works with clean forms of source and target tokens, as shown in the first block. The second source word ‘**anderen**’ is randomly selected to be substituted with its noisy version ‘*andare*’. In *FT*, a source sequence including this noisy form (or its preprocessed version) is sent to the translation engine but the target sequence remains untouched. *TAFT* works with a slightly different data format where the source sequence includes the noisy input and at the same time its clean version (namely ‘**anderen**’) is appended to the original target sequence. With this simple technique, the NMT model is forced to generate translations, spot noisy source tokens, and correct them all together. This could be considered as an implicit form of multi-tasking without changing the neural architecture.

<i>Original</i>
alle anderen waren anderer meinung . all of the others were of a different opinion .
<i>FT</i>
alle and@@ are waren anderer meinung . all of the others were of a different opinion .
<i>TAFT</i>
alle and@@ are waren anderer meinung . all of the others were of a different opinion . anderen

Table 1: How *FT* and *TAFT* process training examples. Noise is added to the boldfaced word. This example is selected from our German→English corpus. In each block, the first sequence is from the source and the second sequence is from the target side. Sequences are pre-processed and tokenized.

The fusion of translation and correction knowledge on the decoder side seems to be useful (see our experimental results in Tables 3 and 4).

It should be noted that at test time in *TAFT*, the engine only generates tokens of the target sequence, i.e. it stops decoding as soon as it visits the end of the target sequence. Generating target tokens together with clean source words is a training-time technique to improve the robustness of the model. Therefore, this extension does not slow down the model or change anything about it at inference. Moreover, if any segmentation scheme such as byte-pair encoding (*bpe*) (Sennrich et al., 2016) is applied to input words during preprocessing, the noisy form also needs to be preprocessed accordingly. The same rule applies to the clean form appended to the target sequence too, namely it needs to follow the segmentation scheme of the target side. In Table 1, the noisy form ‘*andare*’ is segmented into ‘*and@@*’ and ‘*are*’ via the source-side *bpe* model and the correct form ‘**anderen**’ is appended as is because the target *bpe* model was able to recognize it as an existing entry.

3.2 Controlled Denoising

FT and *TAFT* have no control over the encoder’s output, and it is assumed that the decoder alone is powerful enough to handle representations of noisy inputs and deliver correct translations. This assumption might fail in practice, so we place a filter after the encoder to purify source representations before sending them to the decoder. We refer to this process as *Controlled Denoising (CD)*.

The idea behind *CD* is to force the encoder to correct its noisy representations. To implement this mechanism, we accompany the main encoder (the one that is connected to the decoder) with an auxil-

ary, pre-trained encoder. These two encoders have an identical architecture and work with the same vocabulary. The main encoder consumes sequences that may include noisy tokens but the auxiliary one is always fed with clean sequences. These two encoders meet before the first layer of the decoder to ensure they both generate the same representations regardless of any discrepancies that may occur in their input. If there appears a noisy token in the input of the main encoder its output would differ from that of the auxiliary one. Therefore, we match the output of these two encoders via a loss function to ensure the main encoder is able to handle/ignore the input noise.

In the absence of noise, the main encoder mimics the auxiliary encoder, but when noise is added the main encoder’s outputs may deviate from expected ones. The loss function in between the two encoders helps the main encoder correct itself and push its outputs closer to clean representations (auxiliary encoder’s outputs). This architecture is illustrated in Figure 1.

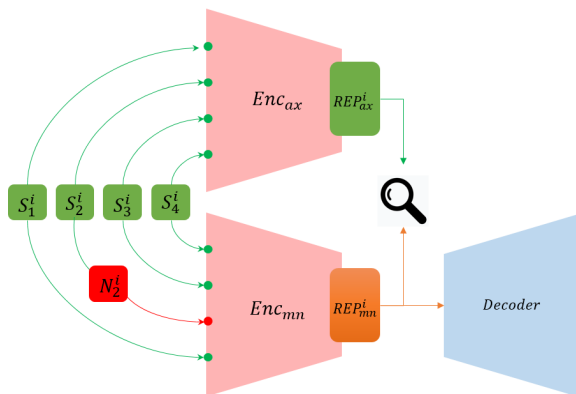


Figure 1: The *CD* architecture. S_j^i is the j -th token of the i -th sequence whose perturbed form is N_j^i . REP_{mn} and REP_{ax} show the representations of the input sequences generated by the *main* and *auxiliary* encoders, respectively. These two representations are compared to each other via a loss function (denoted with \mathbf{Q}) to ensure that the main encoder is able to handle noisy inputs.

Conventional recurrent and convolutional encoders usually squeeze the representation of the input sequence into a single vector, but Transformers due to their non-recurrent architecture perform differently and instead generate s vectors if the input sequence consists of s tokens. This makes the comparison between outputs of the main and auxiliary encoders challenging. Because, when an input sequence is perturbed with noise the length of

the noisy sequence could vary from the clean one, e.g. one token can be added/dropped or the noisy token can be decomposed into multiple units via *bpe* (as shown in Table 1). In such cases, the shape of encoders’ outputs does not even match and a vector-to-vector comparison is impossible. To handle these issues, we learn a dedicated representation for *the entire input sequence*, so comparing outputs would be straightforward. We do this form of sequence modelling by following the same idea proposed for the CLS token in Devlin et al. (2019). We refer to this sequence-level representation as REP in our setting.

In Figure 1, the inputs to the main and auxiliary encoders are $[S_1^i, N_2^i, S_3^i, S_4^i]$ and $[S_1^i, S_2^i, S_3^i, S_4^i]$, respectively, and their sequence-level representations are REP_{mn} and REP_{ax} . If our Transformer encoder is fed with s tokens it returns $s + 1$ vectors with the last one being REP . This token is only used for comparison purposes between encoders and is not sent to the decoder.

To train our models with *CD*, we extend the original translation loss (\mathcal{L}_{tr}) with an additional term, \mathcal{L}_{CD} , as defined in Equation 1:

$$\mathcal{L} = \alpha\mathcal{L}_{tr} + \beta\mathcal{L}_{CD}$$

$$\mathcal{L}_{CD} = \sum_i \text{MSE}(REP_{mn}^i, REP_{ax}^i) \quad (1)$$

where $\text{MSE}()$ is the mean-square error and REP_{mn}^i and REP_{ax}^i are the sequence-level representations of the i -th training sequence generated by our two encoders. α and β are weights to adjust the contribution of each loss function in the training process. Check Section 4 for their values.

As previously mentioned, the auxiliary encoder is a pre-trained model (trained on the clean/original sentences of the same dataset with an identical vocabulary set and architecture as the main encoder) and is only used to generate reference representations for the main encoder, so its parameters are not updated during training and only main encoder and decoder’s parameters are impacted in the back-propagation phase. It should also be noted that the auxiliary encoder is used during training and is decoupled later for inference. Therefore, the size of the final model and inference time remain the same as in the original Transformer.

3.3 Dual-Channel Decoding

This approach relies on the idea of data augmentation and multi-tasking, and tries to break down

noise-robust NMT into two tasks of *denoising* and *translation*. Our *DCD* architecture has two decoder channels: One for generating the rectified noisy tokens (D_{dn}) detected in the input sequence and the other one (D_{tr}) for actual translations. The extra task defined in addition to translation is meant to guide the decoder by providing richer information and make it robust against input noise. This architecture is illustrated in Figure 2.

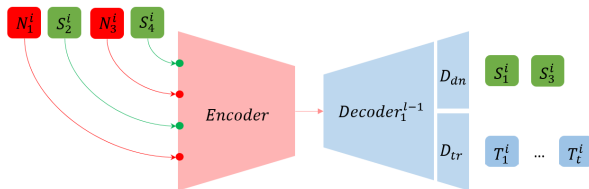


Figure 2: The DCD architecture. D_{dn} generates $[S_1^i, S_3^i]$ which are the clean version of noisy tokens on the input side and D_{tr} samples t tokens from the target vocabulary to generate the final translation $[T_1^i, \dots, T_t^i]$. The decoder has l layers where $(l - 1)$ of them are shared.

The original Transformer decoder has l layers. In our *DCD* extension, the first $l - 1$ layers are shared in between two tasks, but the last layer has dedicated components for each. D_{dn} is a unique decoder layer that is responsible to generate clean forms of any noisy token that can appear in the input, e.g. in Figure 2 the first (N_1^i) and third (N_3^i) tokens are perturbed so D_{dn} generates S_1^i and S_3^i (their clean versions) as its output. D_{dn} is trained via a dedicated loss function (\mathcal{L}_{dn}) designed for this task. On the other side, D_{tr} is another decoder layer that shares no parameter with D_{dn} . This layer is placed over the decoder’s lexicon and samples target words to generate the final translation. This layer is connected to \mathcal{L}_{tr} to penalize incorrect translations. In this setting, the final loss function is a composition of two losses, as defined in Equation 2:

$$\mathcal{L} = \alpha\mathcal{L}_{tr} + \beta\mathcal{L}_{dn} \quad (2)$$

The main purpose of having such a semi-shared architecture for each task is to benefit from the power of multi-tasking. Both D_{tr} and D_{dn} are triggered with a mixture of information about translation and denoising provided by the first $l - 1$ layers of the decoder; then they use their dedicated modules/parameters to generate different outputs for their particular task. Similar to *CD*, this technique is also employed during training and at inference we do not require D_{dn} . This should ensure similar

memory consumption and inference speed as the vanilla Transformer model.

4 Experimental Study

Datasets To evaluate our models, we trained engines to translate the English–German (En–De) pair in both directions. In the interest of fair comparisons, we used the same datasets as the original Transformer (Vaswani et al., 2017), so our training set is the WMT–14 dataset² with 4.5M parallel sentences and for development and test sets we used newstest–13 and newstest–14, respectively.

Nowadays, almost all state-of-the-art NMT models rely on subwords. We also followed the same tradition and preprocessed the target side of our datasets with *bpe* (Sennrich et al., 2016). For the source side, we used different segmentation schemes with different granularities as we are studying the impact of noisy inputs. Our source tokens can appear in surface forms (words) or can be segmented into *bpe* tokens or even characters. We refer to these settings as *word2bpe*, *bpe2bpe*, and *char2bpe*, respectively.

The size of the lexicon generated by our *bpe* model also matches the setting proposed for the original Transformer model (Vaswani et al., 2017). For the *word2bpe* setting, we keep the top 48K frequent words for each English and German sides and ignore the rest by substituting with a special UNK token. This configuration is learned through an empirical study to maximize translation quality.

Hyper-parameters We carried out multiple experiments to study how each of *word2bpe*, *bpe2bpe*, and *char2bpe* settings react in conjunction with our models and what values should be used for hyper-parameters.

In these experiments, we did not change the neural architecture for the *FT* and *TAF* models and only trained translation engines with augmented datasets. We realized that fine-tuning can be improved if we slightly change the learning-rate scheduling. The original Transformer uses the Noam scheduling (Vaswani et al., 2017) that employs a linear warm-up strategy followed by a decaying function. We changed it to a simple exponential staircase decay with an initial learning rate of 0.001 and a decay rate of 0.5 after every 5,000

²<http://statmt.org/wmt14/translation-task.html>

steps.

As our translation engines are already trained and we only need to fine-tune them, we can ignore the warm-up strategy. We can start from a non-zero value and carefully decrease the learning rate until models converge. We fine-tune all our models for 50,000 steps with this scheduler. Figure 3 illustrates the difference between Noam and our scheduler.

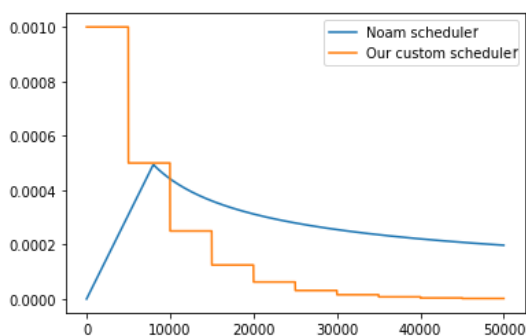


Figure 3: Comparing Noam to our custom learning rate scheduler. The y axis represents the learning rate and the x axis shows the number of steps.

Apart from *char2bpe* that uses the batch size of 12,288, all other settings process 4,096 tokens in each batch. *char2bpe* is a character-based model and the more characters it processes, the better performance it gains. We set α and β (loss weights) to [0.75, 0.25] and [0.9, 0.1] for *CD* and *DCD*, respectively. Our models are trained using Adam (Kingma and Ba, 2014). If the value of any other hyper-parameter (such as the embedding dimension etc) is not clearly mentioned in the paper that means we use the original value of it proposed by Vaswani et al. (2017).

4.1 Introducing Noise

To train/test our engines, we need to perturb source sentences by injecting noise. A noisy word can be created by adding, dropping, or replacing a character in a word or by imposing any other deformations (Cheng et al., 2018; Michel and Neubig, 2018). However, all these techniques artificially produce new forms that might not necessarily reflect real-world noise. We thus use a particular type of noise which is known as *natural noise* in the literature (Belinkov and Bisk, 2018). This form is an error that can naturally appear in any text. Researchers collected lists³ of frequently-occurring

³<https://github.com/ybisk/charNMT-noise/tree/master/noise>

mistakes/typos in different languages from existing corpora and made them available. In our experiments, we randomly pick a candidate word and retrieve its noisy version from the aforementioned lists. This way we could ensure that our noisy dataset is representative of what we may encounter in real life.

To create our training sets, we randomly select 50% of *sentences* to perturb with noise. We only destroy one word in each sentence. Noise is added to surface forms, so if the neural encoder is designed to work with a different granularity, all necessary preprocessing steps are applied accordingly, e.g. in Table 1, first the candidate word (**anderen**) is perturbed (*andare*), then *bpe* is applied to the noisy form to have a consistent input (*and@@are*) with the encoder’s vocabulary.

To add noise to our test sets we have a slightly different and relatively aggressive approach. We are interested in challenging our models to see if they can tolerate high volumes of noise, so we created 4 noisy test sets in which 5%, 10%, 20%, and 30% of entire *words* (not sentences) are destroyed. Adding noise based on the percentage of words instead of sentences makes translation quite challenging because perturbing for example 10% of sentences (with one noisy word) in our 3003-sentence test set only generates 300 noisy words whereas this number would be around 7000 if we perturb 10% of the entire words. Unlike the training setting where we only perturb one word in a sentence, in the test setting, multiple words can be impacted.

Since, this is the first work (to the best of our knowledge) that particularly studies Transformers for their ability to tackled noise we only selected to work with *natural* noise, which seems to be the most realistic form. However, our work can be extended by investigating the impact of other famous noise classes such as *Swap*, *Mid*, *Rand* etc. For detailed information on noise classes see Belinkov and Bisk (2018), Khayrallah and Koehn (2018), and Michel and Neubig (2018).

4.2 Baseline Models

As our baseline, we trained a Transformer with a slight modification in its architecture. Kasai et al. (2020) conducted research and showed that the number of encoder and decoder layers do not necessarily need to match and we can have imbalanced Transformers with deep(er) encoders and

		0%	5%	10%	20%	30%
En→De	<i>word2bpe</i>	28.48	22.21 (-22%)	17.05 (-40%)	10.28 (-64%)	5.99 (-79%)
	<i>bpe2bpe</i>	28.46	24.82 (-13%)	21.58 (-24%)	15.98 (-44%)	11.89 (-58%)
	<i>char2bpe</i>	26.07	24.23 (-7%)	21.84 (-16%)	18.37 (-30%)	15.01 (-42%)
	<i>ConvT</i>	25.46	22.55 (-11%)	20.13 (-21%)	14.9 (-41%)	11.29 (-56%)
De→En	<i>word2bpe</i>	25.94	23.28 (-10%)	20.32 (-22%)	15.79 (-39%)	12.00 (-54%)
	<i>bpe2bpe</i>	28.04	24.87 (-11%)	21.61 (-23%)	16.11 (-43%)	11.48 (-59%)
	<i>char2bpe</i>	26.59	25.01 (-6%)	22.73 (-15%)	19.42 (-27%)	15.93 (-40%)
	<i>ConvT</i>	27.08	24.01 (-11%)	21.39 (-21%)	16.44 (-39%)	11.59 (-57%)

Table 2: BLEU scores of baseline models. Numbers inside parentheses show how much noise impacts models’ performance. The first row shows the percentage of perturbed test words. All our encoders and decoders have 8 and 4 layers, respectively. The *char2bpe* models consumes one character at a time. The *ConvT* model consumes one word at each step but applies a convolutional operation over all characters of the word before feeding it to the first encoder layer.

shallow(er) decoders. Inspired by that work, we increased the number of encoder layers⁴ from 6 to 8 and decreased the number of decoder layers from 6 to 4. Our Transformer still has 12 layers in total, but the encoder is more powerful which is favourable in our scenario. Noise appears on the source side and we require better encoders to tackle this. Based on our experiments, the 8–4 configuration (\mathcal{T}_4^8) is able to handle noise better than the 6–6 version and all other variants. \mathcal{T}_4^8 is our baseline for all experiments and our other novel architectures are also implemented based on the 8–4 setting.

Belinkov and Bisk (2018) and Karpukhin et al. (2019) used a convolutional, character-based encoder and showed that this improves the robustness of NMT models. They tested this configuration with relatively small datasets or recurrent architectures. We adapted the same idea and equipped the Transformer model with the same convolutional module. This model, which is referred to as *ConvT* in this paper, is another baseline for our experiments. Similar to Karpukhin et al. (2019), character embeddings have 256 dimensions and the convolutional module follows the specifications of Kim et al. (2016). Table 2 summarizes our baseline results. We use the BLEU metric (Papineni et al., 2002) to compare our models. All scores are calculated on detokenized outputs using SacreBLEU⁵ (Post, 2018).

As the table shows, no matter how powerful the engine is, adding even 5% noise is enough to break

the model. Each segmentation scheme shows a unique behaviour. We were expecting a significant deterioration in the *word2bpe* case but for both directions it provides relatively competitive results. *bpe2bpe* seems to be the best as it gives the highest baseline where no noise is involved and shows less drop for noisy test sets. *char2bpe* has the least decline when noise is added but it should be noted that it is not able to compete with others in the absence of noise. Although its degradation is minimal, it degrades from a non-optimal baseline. *ConvT*, despite its sophisticated architecture, could not outperform *bpe2bpe* and this was expected as tuning such a network over our (relatively) large dataset (wmt-14 with 4.5M samples) could be challenging.

4.3 Results for Proposed Models

In this section we report results for *word2bpe*, *bpe2bpe*, and *char2bpe* settings when used with our solutions (*TAFT*, *CD*, and *DCD*).

Table 3 summarizes results related to *word2bpe*. When translating into German, *DCD* outperforms all other models where up to 10% noise is added to the test set. For extreme cases with 20% and 30% noise, *FT* is more effective. In the opposite direction, *FT* and our *TAFT* extension provide the best performance. *TAFT* also shows a very promising result for the 30% test set and even defeats the noise-free setting. The huge gap between the vanilla \mathcal{T}_4^8 and engines equipped with our techniques shows the necessity of building noise-robust NMT models, specially if they are supposed to be deployed in real-world applications.

Results for *bpe2bpe* models are reported in Table 4. For the En→De direction, *CD* is superior

⁴The original Transformer architecture (Vaswani et al., 2017) proposes 6 encoder and 6 decoder layers.

⁵<https://github.com/mjpost/sacrebleu>

Model	0%	5%	10%	20%	30%
En→De					
\mathcal{T}_4^8	28.48	22.21	17.05	10.28	5.99
<i>FT</i>	29.21	27.15	25.32	21.93	17.79
<i>TAFT</i>	29.47	27.33	25.35	21.33	17.29
<i>CD</i>	29.03	27.02	25.00	20.70	16.75
<i>DCD</i>	29.48	27.52	25.65	21.68	17.76
De→En					
\mathcal{T}_4^8	25.94	23.28	20.32	15.79	12.00
<i>FT</i>	27.16	26.96	26.69	26.16	25.83
<i>TAFT</i>	27.00	26.87	26.84	26.27	26.08
<i>CD</i>	27.1	26.83	26.61	26.04	25.57
<i>DCD</i>	27.06	26.86	26.83	26.13	26.03

Table 3: *word2bpe* results. \mathcal{T}_4^8 is a Transformer with 8 encoder and 4 decoder layers. Boldfaced numbers are the best scores of each column.

for all test sets, and this indicates that a loss function over a *bpe*-based encoder could remarkably increase robustness. We observe a similar trend in the previous experiment for the De→En direction. *DCD* is also quite successful when test sets are fairly noisy. It seems *bpe*-based Transformers benefit a lot from multi-tasking since both *TAFT* and *DCD* force the decoder to perform a second task in addition to translation.

Model	0%	5%	10%	20%	30%
En→De					
\mathcal{T}_4^8	28.46	24.82	21.58	15.98	11.89
<i>FT</i>	28.8	27.95	27.01	24.6	21.84
<i>TAFT</i>	28.96	28.03	26.65	24.02	21.16
<i>CD</i>	29.49	28.51	27.68	25.27	22.63
<i>DCD</i>	28.91	28.02	26.89	24.37	21.47
De→En					
\mathcal{T}_4^8	28.04	24.87	21.61	16.11	11.48
<i>FT</i>	28.46	28.4	28.22	27.83	27.51
<i>TAFT</i>	28.73	28.53	28.51	27.93	27.63
<i>CD</i>	28.52	28.42	28.25	27.84	27.50
<i>DCD</i>	28.65	28.49	28.4	27.98	27.65

Table 4: *bpe2bpe* results.

Finally, we summarize results of *char2bpe* models in Table 5. Trends for this set of experiments are relatively consistent with previous ones. For the En→De direction, the best result on average is delivered by *CD* but *DCD* also shows comparable performance. For the opposite direction, *TAFT* and

Model	0%	5%	10%	20%	30%
En→De					
\mathcal{T}_4^8	26.07	24.23	21.84	18.37	15.01
<i>ConvT</i>	25.46	22.55	20.13	14.9	11.29
<i>FT</i>	27.24	26.50	25.92	24.51	23.36
<i>TAFT</i>	27.11	26.41	25.56	23.90	22.14
<i>CD</i>	27.29	26.5	26.05	24.71	23.37
<i>DCD</i>	27.2	26.73	25.88	24.58	23.12
De→En					
\mathcal{T}_4^8	26.59	25.01	22.73	19.42	15.93
<i>ConvT</i>	27.08	24.01	21.39	16.44	11.59
<i>FT</i>	27.31	27.14	27.07	26.83	26.49
<i>TAFT</i>	27.64	27.52	27.32	26.95	26.53
<i>CD</i>	27.26	27.15	26.89	26.78	26.4
<i>DCD</i>	27.71	27.52	27.45	27.06	26.78

Table 5: *char2bpe* results. *ConvT* is added as additional baseline as encoders rely on characters.

DCD are better choices and multi-tasking again shows its impact. Because *char2bpe* is a character-based model we also added results from *ConvT* as another baseline to study if the convolutional operation can mitigate the problem and handle noise better. Experimental results demonstrate that there is no need for such a complex configuration and our techniques can train high-quality engines.

5 Conclusion and Future work

In this paper, we studied the problem of noise in the context of NMT and particularly focused on Transformers. We proposed three novel techniques to augment data and change the training procedure as well as the neural architecture. Experimental results show that our techniques can protect NMT engines from noise. Our models only affect the training phase and do not add any overhead in terms of space and/or time complexities at inference time. The findings of our research can be summarized as follows:

- There is no clear winner among our proposed models. Each approach has its own strength and should be adapted with respect to the problem.
- *FT* and *TAFT* are data-driven techniques and can be applied to existing translation models with minimal effort.
- *CD* and *DCD* require some modifications in

the neural architecture but they are able to provide promising results.

- Multi-tasking was quite useful in this scenario and it seems Transformers benefit a lot when their decoder is informed about source-side noise.

In this research, we ran an extensive number of experiments in order to find the best configuration of each model and optimize hyper-parameters, but there still exist some unexplored topics/areas. In our future work, we are planning to experiment with other language pairs with different morphological and grammatical structures. We are also interested in studying other noise classes. We could only afford to work with one class and we selected natural noise as we find it more realistic among others, but this work can be extended to other noise classes. Finally, our models are not unique to Transformer and NMT. We aim to evaluate them in other language processing/understanding tasks with other architectures.

Acknowledgements

We would like to thank our anonymous reviewers as well as Mehdi Rezagholizadeh and Yimeng Wu for their valuable comments and feedback.

References

- Yonatan Belinkov and Yonatan Bisk. 2018. [Synthetic and natural noise both break neural machine translation](#). In *International Conference on Learning Representations*.
- Alexandre Berard, Ioan Calapodescu, and Claude Roux. 2019. Naver labs Europe’s systems for the WMT19 machine translation robustness task. In *Proceedings of the Fourth Conference on Machine Translation*. Association for Computational Linguistics.
- Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. [Robust neural machine translation with doubly adversarial inputs](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4324–4333, Florence, Italy. Association for Computational Linguistics.
- Yong Cheng, Zhaopeng Tu, Fandong Meng, Junjie Zhai, and Yang Liu. 2018. [Towards robust neural machine translation](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1756–1766, Melbourne, Australia. Association for Computational Linguistics.
- Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. [On the properties of neural machine translation: Encoder–decoder approaches](#). In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. [Learning phrase representations using RNN encoder–decoder for statistical machine translation](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Raj Dabre and Eiichiro Sumita. 2019. [NICT’s supervised neural machine translation systems for the WMT19 translation robustness task](#). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 533–536, Florence, Italy. Association for Computational Linguistics.
- J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.
- Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. 2017. [A convolutional encoder model for neural machine translation](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada. Association for Computational Linguistics.
- Jindřich Helcl, Jindřich Libovický, and Martin Popel. 2019. CUNI system for the WMT19 robustness task. In *Proceedings of the Fourth Conference on Machine Translation*. Association for Computational Linguistics.
- Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Vladimir Karpukhin, Omer Levy, Jacob Eisenstein, and Marjan Ghazvininejad. 2019. [Training on synthetic noise improves robustness to natural noise in machine translation](#). In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 42–47, Hong Kong, China. Association for Computational Linguistics.
- Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*.

- Huda Khayrallah and Philipp Koehn. 2018. On the impact of various types of noise on neural machine translation. In *Proceedings of the Second Workshop on Neural Machine Translation and Generation*, Melbourne. Association for Computational Linguistics.
- Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Xian Li, Paul Michel, Antonios Anastasopoulos, Yonatan Belinkov, Nadir Durrani, Orhan Firat, Philipp Koehn, Graham Neubig, Juan Pino, and Hassan Sajjad. 2019. Findings of the first shared task on machine translation robustness. *arXiv preprint arXiv:1906.11943*.
- Paul Michel and Graham Neubig. 2018. **MTNT: A testbed for machine translation of noisy text**. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 543–553, Brussels, Belgium. Association for Computational Linguistics.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.
- Matt Post. 2018. **A call for clarity in reporting BLEU scores**. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.
- Keisuke Sakaguchi, Kevin Duh, Matt Post, and Ben Van Durme. 2017. Robust word recognition via semi-character recurrent neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2017)*.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.
- Yifu Sun and Haoming Jiang. 2019. **Contextual text denoising with masked language model**. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 286–290, Hong Kong, China. Association for Computational Linguistics.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. **Attention is all you need**. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.
- Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. 2020. A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv:2002.07526*.
- Shuyan Zhou, Xiangkai Zeng, Yingqi Zhou, Antonios Anastasopoulos, and Graham Neubig. 2019. **Improving robustness of neural machine translation with multi-task learning**. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 565–571, Florence, Italy. Association for Computational Linguistics.