Learning to Segment Inputs for NMT Favors Character-Level Processing

Julia Kreutzer[*], *Artem Sokolov*[◇,*]

[*]Computational Linguistics, Heidelberg University, Germany
[◇]Amazon Research, Germany
kreutzer@cl.uni-heidelberg.de, artemsok@amazon.com

Abstract

Most modern neural machine translation (NMT) systems rely on presegmented inputs. Segmentation granularity importantly determines the input and output sequence lengths, hence the modeling depth, and source and target vocabularies, which in turn determine model size, computational costs of softmax normalization, and handling of out-of-vocabulary words. However, the current practice is to use static, heuristic-based segmentations that are fixed before NMT training. This begs the question whether the chosen segmentation is optimal for the translation task. To overcome suboptimal segmentation choices, we present an algorithm for dynamic segmentation, that is trainable end-to-end and driven by the NMT objective. In an evaluation on four translation tasks we found that, given the freedom to navigate between different segmentation levels, the model prefers to operate on (almost) character level, providing support for purely character-level NMT models from a novel angle.

1. Introduction

Segmentation of input sequences is an essential preprocessing step for neural machine translation (NMT) and has been found to have a high positive impact on translation quality in recent WMT shared task evaluations [1, 2]. This success can be explained statistically, since shorter segments are beneficial for reducing sparsity: They lower the type-to-token ratio, decrease the number of out-of-vocabulary (OOV) tokens and singletons, which improves the coverage of unseen inputs.

Two subword segmentation methods are presently the state-of-the-art in NMT: the *byte-pair encoding* (BPE), that starts with a dictionary of single characters and iteratively creates a new entry from the two currently most frequent entries [3, 4], and a similar *wordpiece* (WP) model [5].

While being empirically more successful than word-based NMT, both BPE and WP are preprocessing heuristics, they do not account for the translation task or the language pairs at hand (unless applied to both sides jointly), and require additional preprocessing for languages that lack explicit word separation in writing. Being used in a pipeline fashion, they make it impossible for an NMT system to resegment an unfavorably presplit input and require consistent application of the same segmentation model during testing, which adds an integration overhead and contributes to the 'pipeline jungles' in production environments [6].

On the other extreme from word-based NMT models lie purely character models. Their advantages are smaller vocabularies, thus smaller embedding and output layers, allowing for more learning iterations within a training time budget to improve generalization [7], and no preprocessing requirements. At the same time, longer input sequences aggravate known optimization problems with very large depths of time-unrolled RNNs [8] and may require additional memory for tracking gradients along the unrolling steps.

In this work[1], we pose the following question: **what would the input segmentations look like if the NMT model could decide on them dynamically?** Instead of heuristically committing to a fixed (sub)word- or character-segmentation level prior to NMT training, this would allow segmentation for each input to be driven by the training objective and avoid solving the trade-offs of different levels by trial and error. To answer this question, we endow an NMT model with the capacity of adaptive segmentation by replacing the conventional lookup embedding layer with a 'smart embedding' layer that sequentially reads input characters and dynamically decides to group a block of them into an output embedding vector, feeding it to the upstream NMT encoder before continuing with the next block (with an optional reverse process on the target side). To signal that a block of characters, encoded as an embedding vector, is ready to be fed upstream, we use accumulated values of a scalar halting unit [9], which learns when to output this block's embedding. It simultaneously affects weighting probabilities of intermediate output vectors that compose the output embedding. Thanks to this weighting, our model is fully differentiable and can be trained end-to-end. Similarly to BPE, it has a hyper-parameter that influences segmentation granularity, but in contrast to BPE this hyper-parameter does not affect the model size. While we evaluate our on-the-fly segmentation algorithm on RNN-based NMT systems, it is transferable to other NMT architectures (CNN [10] or Transformer [11]), since it only replaces the input embedding layer. Empirically, we find a strong preference of such NMT models to operate on segments that are

Work was done while the first author was interning at Amazon, Berlin.

[1]Extended report can be found at arxiv.org/abs/YYMM.NNNNN.

only one to a few characters long. This turns out to be a reasonable choice, as in our experiments character-level NMT systems of smaller or comparable size were able to outperform word- and subword-based systems, which corroborates results of [12, 13]. Given this finding and the unique advantages of character-level processing (no pipelining, no tokenization, no additional hyperparameters, tiny vocabulary and memory, and robustness to spelling errors [14]), we hope that character-level NMT, and in general character-level sequence-to-sequence learning, will receive more attention from researchers.

Note that, although our character-based models outperform (sub)word-based ones with similar architectures on some datasets, we are not seeking to establish a new state-of-the-art in NMT with our model. Our goal is to isolate the effects of segmentation on quality by introducing a flexibility-enhancing research tool. Therefore, in the comparisons between (sub)word- and character-based models we purposely avoided introducing changes to our baseline RNN NMT architecture beyond upgrading the embedding layer.

2. Related Work

To tackle the OOV problem in word-level models, [15] proposed a hybrid model that composes unknown words from characters both on encoder and decoder side. While their approach relies on given word boundaries, they report a purely character-based baseline performing as well as a word-based model with unknown word replacement, but taking 3 months to train, which seems to have cooled off the NMT community in investigating fully character-based models as an alternative to (sub)word-based ones. Unlike [15], we found that despite the training speed being slower than for (sub)word vocabularies, it is possible to train reasonable character-level models within a few weeks. To combine the best of both worlds, [16] proposed hierarchical en-/decoders that receive inputs on both word- and character-level. The encoder learns a weighted recurrent representation of each word's characters and the decoder receives the previous target word and predicts characters until a delimiter is produced. Similar to our work, they find improvements over BPE models. The idea to learn composite representations of blocks of characters is similar to ours, but their approach requires given word boundaries, which our model learns on-the-fly. [12] combined a standard subword-level encoder with a two-layer, hierarchical character-level decoder. The decoder has gating units that regulate the influence of the lower-level layer to the higher-level one, hence fulfilling a similar purpose as our halting unit. This model outperforms a subword-level NMT system, and achieves state-of-the-art on a subset of WMT evaluation tasks. While not requiring explicit segmentation on the target side, the model still relies on given source segmentations. Finally, [14] proposed a fully character-level NMT model. They mainly address training speed, which [15] identified as a problem, and introduce a low-level convolutional layer over character embeddings to extract information from variable-length character n-grams for higher-level processing with standard RNN layers. Thus, overlapping segments are modelled with a length depending on the filters.

Perhaps closest to our work is [13], where each layer of a hierarchical RNN encoder is updated at different rates, with the first layer modelling character-level structures, the following modelling sub(word)-level structures. They introduce a binary boundary detector, similar to our halting unit, that triggers feeding of a representation to the next level, so that latent hierarchical structures without explicit boundary information are learnt. Unlike our fully-differentiable model, such discrete decisions of the boundary detector prohibit end-to-end differentiability, forcing a recourse to the biased straight-through estimator [17]. On the other hand, while our model relies on a to-be-tuned computation time penalty, [13] do not impose constraints on the number of boundaries.

3. Jointly Learning to Segment and Translate

Instead of committing to a single segmentation before NMT model training, we propose to learn the segmentation-governing parameters along with the usual network parameters in a end-to-end differentiable manner. With this approach, we get rid of pipelining and pre-/postprocessing, and can adaptively segment arbitrary inputs we encounter during training or testing. Our segmentations are context-dependent, i.e. the same substring can be segmented into different parts in different contexts. Being able to smoothly interpolate between word-based and character-based models we allow the model to find a sweet spot in between.

We extend the *Adaptive Computation Time* (ACT) paradigm [9], where a general RNN model is augmented with a scalar halting unit that decides how many recurrent computations are spent on each input. For segmentation, we use the halting unit to decide how many inputs (characters) a segment consists of. The output of the ACT module can thus be thought of as an 'embedding' vector for a segment that replaces the classic lookup embedding for (sub)words in standard NMT models. While our model can in principle use larger units as elementary inputs, we will focus on character inputs to be able to model the composition of arbitrary segments. That means that we only add a small amount of parameters to a basic character-based model, but explicitly model higher-level merges of characters into subwords.

3.1. ACT for Dynamic Depth

Here we summarize the ACT model [9]. It is applicable to any recurrent architecture that transforms an input sequence $\mathbf{x} = (x_1, \ldots, x_T)$ into outputs $\mathbf{o} = (\bar{o}_1, \ldots, \bar{o}_T)$ via computing a sequence of states $\mathbf{s} = (s_1, \ldots, s_T)$ through a state transition function $\mathcal{S}$ on an embedded input Ex_t and a linear output projection defined by matrix W_o and bias b_o:

$$s_t = \mathcal{S}(s_{t-1}, Ex_t), \qquad o_t = W_o s_t + b_o \tag{1}$$

Instead of stacking multiple RNN layers in $\mathcal{S}$ to achieve increased complexity of an RNN network, the ACT model

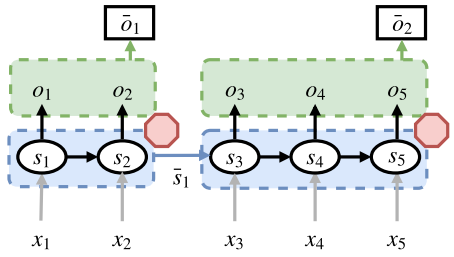

Figure 1: Diagram of the ACT-ENC encoder. Note the differences to the original ACT model: An input is here read on *every* internal recurrent iteration (gray arrows) and the halting unit (red stop sign) is repurposed to trigger feeding of an encoded embedding vector of a block of characters to the upstream NMT layers.

dynamically decides on the number of necessary recurrent steps (layers) for every input x_t. This saves computation on easy inputs, while still being able to use all of the processing power on hard inputs before emitting outputs. Concretely, an ACT cell performs an arbitrary number of internal recurrent applications of $\mathcal{S}$ for each input x_t:

$$s_t^n = \begin{cases} \mathcal{S}(\bar{s}_{t-1}, Ex_t), \text{ if } n = 1 \\ \mathcal{S}(s_t^{n-1}, Ex_t), \text{ otherwise} \end{cases} \tag{2}$$

The total number of internal steps is $N(t) = \min\{n' : \sum_{n=1}^{n'} h_t^n \geq 1 - \epsilon\}$, where $\epsilon \ll 1$ and h_t^n is the scalar output of sigmoid halting unit,

$$h_t^n = \sigma(W_h s_t^n + b_h). \tag{3}$$

Once halted, the final output $\bar{o}_t$ and state $\bar{s}_t$ (which is fed to the next ACT step in (2)) are computed as weighted means of intermediate outputs and states:

$$\bar{s}_t = \sum_{n=1}^{N(t)} p_t^n s_t^n, \qquad \bar{o}_t = \sum_{n=1}^{N(t)} p_t^n o_t^n \tag{4}$$

where probabilities p_t^n are defined as

$$p_t^n = \begin{cases} R(t), \text{ if } n = N(t) \\ h_t^n, \text{ otherwise} \end{cases} \tag{5}$$

and remainders $R(t) = 1 - \sum_{n'=1}^{N(t)-1} h_t^{n'}$. Finally, to prevent the network from pondering on an input for too long, the remainder $R(t)$ is added as a penalty to the RNN training loss (usually cross-entropy (XENT)) with a weight τ:

$$L_{\text{ACT}} = L_{\text{XENT}} + \tau R(t). \tag{6}$$

Thanks to (4), the model is deterministic and differentiable.

3.2. ACT for Dynamic Segmentation

We now describe how to use the ACT paradigm to enhance an encoder for dynamic segmentation on the source side (ACT-ENC). We reuse the idea of halting units, mean field updates and τ-penalized training objective, but instead of learning how much computation is needed for each atomic input, we learn how much computation to allow for an aggregation of atomic inputs, i.e. one segment.

The input to an ACT-ENC cell is a sequence of one-hot-encoded characters $\mathbf{x} = (x_1, \ldots, x_{T_\mathbf{x}})$. The ACT-ENC, depicted in Figure 1, receives one input x_t at a time and decides whether to halt or not. In the case of no halting, the cell proceeds reading more inputs; if it halts, it produces an output 'embedding' $\bar{o}$ of a block of characters read so far, and the cell resets for reading the next block. The sequence of the output embeddings $\mathbf{o} = (\bar{o}_1, \ldots, \bar{o}_{T_\mathbf{o}})$ is then fed to upstream standard (possibly bidirectional) NMT encoder layers, replacing the usual, one-hot encoded, (sub)word lookup embeddings. The length of $\mathbf{o}$ is variable: The more frequently ACT-ENC halts, the more embeddings are generated. In extreme cases, it can generate one embedding per input ($T_\mathbf{o} = T_\mathbf{x}$) or just one embedding for the full sequence of inputs ($T_\mathbf{o} = 1$).

Algorithm 1 ACT-ENC

Input: Weights W_o, b_o, W_h, b_h, transition function $\mathcal{S}$, embeddings E_{src}, inputs $\mathbf{x} = (x_1, \ldots, x_{T_\mathbf{x}})$
Output: Outputs $\mathbf{o} = (\bar{o}_1, \ldots, \bar{o}_{T_\mathbf{o}})$, remainder R

1: $\mathbf{o} = [\,]$ ▷ empty sequence
2: $R = 0, H = 0$ ▷ init remainder and halting sum
3: $\bar{s} = \mathbf{0}, \bar{o} = \mathbf{0}, s_0 = \mathbf{0}$ ▷ init mean state and output
4: **for** $t = 1 \ldots T_\mathbf{x}$ **do** ▷ loop over inputs
5: $s_t = \mathcal{S}(s_{t-1}, E_{src}\, x_t)$ ▷ new state
6: $o_t = W_o s_t + b_o$ ▷ new output
7: $h_t = \sigma(W_h s_t + b_h)$ ▷ halting score
8: $f = [\![H + h_t \geq 1 - \epsilon]\!]$ ▷ halting flag
9: $p_t = (1 - f)\, h_t + f\, (1 - H)$ ▷ halting probability
10: $H = H + h_t$ ▷ update halting sum
11: $\bar{s} = \bar{s} + p_t s_t, \bar{o} = \bar{o} + p_t o_t$ ▷ mean state and output
12: $R = R + (1 - f)\, h_t$ ▷ increment remainder
13: **if** f **then**
14: $\mathbf{o} = \mathbf{o}^\frown[\bar{o}]$ ▷ append output
15: $s_t = \bar{s}$ ▷ overwrite for next step
16: $\bar{s} = \mathbf{0}, \bar{o} = \mathbf{0}, H = 0$
17: $R = (1 - R)/t$ ▷ normalize remainder

In more detail, ACT-ENC implements the pseudocode given in Algorithm 1. Let $\mathcal{S}(s_{t-1}, i_t)$ be any recursive computation function (in this work we use GRUs) of an RNN that receives a hidden state s_{t-1} and an input vector i_t at time step t and computes the new hidden state s_t. In line 5 this function is computed on the regular previous state or, if there was a halt in the previous step (line 13), on the mean state vector $\bar{s}$ that summarizes the states of the previous segment (line 15, cf. (4), 1st eq.). Per-step outputs o_t are computed from the hidden states s_t with a feed-forward layer (line 6, cf. (1), 2nd eq.). A sigmoid halting unit computes a halting score in each step (line 7, cf. (3)). The halting probability for step t is either the halting score h_t or the current value of remainder $1 - H$ to ensure that all halting probabilities within one segment form a distribution (line 9, cf. (5)). ϵ is set to a small number to allow halting after a single step.

Whenever the model decides to halt, an output embedding $\bar{o}$ is computed as a weighted mean of the intermediate outputs of the current segment (line 14, cf. (4), 2nd eq.). The weighted mean on the one hand serves the purpose of circumventing stochastic sampling, on the other hand can be interpreted as a type of intra-attention summarizing the intermediate states and outputs of the segment. The halting scores from each step are accumulated (line 12) to penalize computation time as in (6). The hyperparameter τ here controls the segment length: The higher its value, the more preference will be given to smaller remainders, i.e. shorter segments. We introduce an additional normalization by input length (line 17), such that longer sequences will be allowed more segments than shorter sequences. This implementation exploits the fact that ACT-ENC outputs are weighted means over time steps and updates them incrementally. The algorithm allows efficient minibatch processing by maintaining a halting counter that indicates which embedding each current intermediate output in the batch contributes to. Incremental updates of embeddings and states are achieved with masks depending on the halting position.

4. Experiments

We reimplemented the Groundhog RNN encoder-decoder model with attention [18] in `MxNet Gluon` to allow for dynamic computation graphs. We report results on four language directions and domains, for word-, subword-, character-level and ACT-ENC segmentation: German-to-English TED talks, Chinese-to-English web pages, Japanese-to-English scientific abstracts and French-to-English news. Table 1 gives a data overview.

The IWSLT data is split and processed as in [19]; since it comes pretokenized and lowercased, models are evaluated with tokenized, lowercased BLEU (using `sacrebleu`) and chrF scores on character bigrams [20]. For WMT, we used the 2014 dataset prepared for [18], additionally filtering the training data to include only sequences of a lengths 1 to 60, and models are evaluated with cased BLEU and chrF (`sacrebleu`, with the "13a" tokenizer).

The CASIA and ASPEC data are, respectively, from the 2015 China Workshop on MT (CWMT), used without additional pre-processing, and from the WAT 2017 SmallNMT shared task, pretokenized with WP. Both datasets have BPE and WP vocabularies of around 16k for each side, and we report cased BLEU and chrF on them.

Hyperparameters. All models are trained with Adam [21] and a learning rate of 0.0003, halved whenever the validation score (tokenized BLEU) has not increased for 3 validations. Training stopped when the learning rate has been decreased 10 times in a row. All models use recurrent cells of size 1,000 for the decoder, with a bidirectional encoder of size 500 for each direction, input and output embedding of size 620, and the attention MLP of size 1,000, all following [18]. When multiple encoders layers are used, they are all bidirectional [22] with attention on the uppermost layer. The ACT layer for ACT-ENC models has size 50 for IWSLT, CASIA and ASPEC, and 25 for WMT (picked from $\{25, 50, 75, 100, 150\}$). The word-based models on IWSLT and WMT have a vocabulary of 30k for each side, the BPE models have separate 15k vocabularies for IWSLT and a joint 32k vocabulary for WMT. For IWSLT, CASIA and ASPEC all characters from the training data were included in the vocabularies, resulting in vocabulary sizes of 117 (de) and 97 (en), 7,284 (zh) and 166 (en), and 3,212 (ja) and 233 (en), respectively. For WMT the vocabularies included the 400 most frequent characters on each side. Word- and BPE-based models are trained with minibatches of size 80, character-based models with 40. The maximum sequence length during training is 60 for word- and BPE-based models, 200 for character-based models and 150 for ACT-ENC, to fit into available memory. $\tau = 1.0$ delivered the highest BLEU score for IWSLT and CASIA, $\tau = 0.8$ for WMT and $\tau = 0.7$ for ASPEC. Following [9], we fixed $\epsilon = 0.01$ in all the experiments. During inference, we use beam search with a beam size of 5 and length-normalization.

Data	Domain	Lang	Train	Dev	Test
IWSLT	TED talks	de-en	153,352	6,970	6,750
CASIA	web	zh-en	1,045,000	2,500	2,500
ASPEC	sci. abstracts	ja-en	2,000,000	1,790	1,812
WMT	news	fr-en	12,075,604	6,003	3,003

Table 1: Data statistics (number of parallel sentences).

Data	Model	BLEU	chrF	Param	SegLen	TrainTime
IWSLT de-en	Word	22.11	0.44	80.5M	4.66	23h
	BPE	25.38	0.49	46.5M	4.09	20h
	Char	22.63	0.46	13.4M	1.00	1d22h
	ACT-ENC	22.67	0.46	13.5M	1.88	9d21h
CASIA zh-en	BPE	10.59	0.37	49.9M	1.72	18h
	Char	12.60	0.40	21.0M	1.00	10d6h
	ACT-ENC	9.87	0.36	21.3M	1.006	3d13h
ASPEC ja-en	WP	21.05	0.53	50.0M	2.07	4d4h
	Char	22.75	0.55	15.6M	1.00	24d15h
	ACT-ENC	15.82	0.46	15.6M	1.0007	15d4h
WMT fr-en	Word	20.32	0.49	80.5M	5.19	4d9h
	BPE	27.02	0.55	86.0M	4.05	3d23h
	Char	24.25	0.53	14.1M	1.00	9d
	ACT-ENC	13.74	0.42	14.2M	1.82	13d8h

Table 2: Results on test sets for 1-layer models, and number of parameters and average source segment lengths on dev sets. Time to reach stopping criterion is in (d)ays and (h)ours.

Evaluation Results. Table 2 lists the results for the most comparable, 1-layer, configuration. BPE/WP models expectedly outperform word-based models, however word-based models are also outperformed by character-based models. The picture is similar w.r.t. the chrF with even smaller relative differences. The ACT-ENC model with one unidirectional ACT layer manages to match the 1-layer bidirectional character-based model on IWSLT. But it does not reach the results of other models on CASIA and ASPEC, which can be explained by increased complexity of doing simultaneous segmentation during training on sentences longer than the average sentence length in IWSLT. However, the main

Data	Model	BLEU	chrF	Param	TrainTime
IWSLT de-en	Word, 4-layer	24.54	0.45	97.0M	1d8h
	BPE, 1-layer	25.38	0.49	46.5M	20h
	Char, 5-layer	**28.19**	**0.51**	26.9M	3d10h
	ACT-ENC, 3-layer	25.10	0.49	25.6M	9d7h
CASIA zh-en	BPE, 3-layer	11.01	0.38	58.9M	24h
	Char, 3-layer	**13.43**	**0.42**	30.0M	5d6h
	ACT-ENC, 2-layer	10.35	0.37	21.3M	10d
ASPEC ja-en	WP, 3-layer	22.02	**0.55**	61.4M	4d2h
	Char, 1-layer	**22.75**	**0.55**	15.6M	24d15h
	ACT-ENC, 1-layer	15.82	0.46	15.6M	15d4h
WMT fr-en	Word, 2-layer	21.04	0.48	94.0M	4d16h
	BPE, 3-layer	**27.93**	**0.56**	98.0M	5d3h
	Char, 6-layer	27.23	0.55	27.6M	18d13h
	ACT-ENC, 2-layer	14.01	0.43	21.7M	9d10h

Table 3: Results on respective test sets after tuning the number of encoder layers (from 1 to 6) on the dev set.

finding here is that ACT-ENC recovers an almost character-level segmentation (compare the "SegLen" column in Table 2). On the IWSLT dev set, the average segment length is only 1.88 (with a maximum of 5 chars per segment). For CASIA and ASPEC domains and with the larger datasets than IWSLT, the ACT-ENC segmentations becomes more fine-grained: The average segment length is, respectively, just 1.006 and 1.0007 on the dev set, with a maximum of 2 chars per segment. Given that the character model outperforms the model with the BPE/WP segmentation, it is not surprising that ACT-ENC converged to the character segmentation. We hypothesize that ACT-ENC could not improve over the 1-layer bidirectional character model because of complexity of identifying segments in Chinese and Japanese, uni-directionality of its initial processing layer, and the increased hardness of optimization of character models with lots of non-linearities [23]. Similarly for WMT, failing to exactly match the performance of the character model could be caused by harder optimization task on particularly long sentences in the WMT data, and uni-directionality of ACT-ENC. The ACT-ENC's segment length on the dev set is 1.82, with a maximum of 6 chars per segment, again close on average to a purely character segmentation.

Inspired by the ACT-ENC's recovery of almost character segmentation and by the competitive performance of pure character-based models, we decided to verify if the advantage of character-level processing carries over to multiple layers. Since the character models are much smaller than their word-/BPE-based counterparts, one should allow multiple layers (consuming the same or less memory) to make up for the difference in number of parameters for fairer comparison. This also aimed to verify whether an increased number of non-linearities (one of ACT's benefits [24]) plays a role.

Table 3 shows the test results after tuning the number of bidirectional encoder layers, from 1 to 6, on dev sets. First, we observe the modest parameter number of character models even with multiple layers, that allows them to take advantage of deeper cascades of non-linearities while staying well below the memory budget of (sub)word-based 1-layer models. Second, we discover that BPE/WP models are outperformed by character-based models with multiple encoder layers, achieving gains of 2.8 BLEU points on IWSLT, 0.7 on ASPEC, and losing only 0.7 on WMT (with a minor decrease in chrF), despite having at least 3.5 times fewer parameters. Such ranking of character- and BPE-based models on WMT might be explained by much longer sentences in the corpus, compared to IWSLT and ASPEC, since the ability of character and ACT-based models to cover unseen input is limited by the maximum training sequence length limit (here 200 characters), which on WMT data crops 30.5% of sentences.

Analysis of Segmentation and Outputs. Randomly selected translation examples from the IWSLT dev set and their segmented sources are given in Table 4. In general, when encountering rare inputs, word-based models fail by producing the unknown word token (<unk>), and the BPE-model is able to translate only a more common part of German compounds (e.g. 'tiere' → 'animals'). The character-based models invent words ('altients', 'jes lag') that are similar to strings that they saw during training and the source. In a few cases they fallback to a language-modeling regime having attended to the first characters of a corresponding source word: e.g., instead of translating 'reisen' to 'journeys', the ACT-ENC model translates it to 'rows' (confusing 'reisen' to a similarly spelled German 'reihen'), or 'layering' instead of 'shift work' (confusing 'schichten' to the prefix-sharing 'schichtarbeit'). This is confirmed when inspecting attention scores: The model frequently attends to the correct source word, but mainly to the first characters only. Note that ACT-ENC segmentations are context-dependent, e.g. occurrences of 'tiere' are segmented differently.

Table 5 lists the most frequent segments produced by 1-layer ACT-ENC. For IWSLT, we observe that many segments make sense statistically (frequent or rare patterns) and linguistically to some extent: Many of the frequent segments include whitespace (itself a frequent symbol); 2-gram segments amongst others include frequent word suffixes ('en', 'in', 'er'), but also frequent diphthongs ('ei' and 'ie'); 3-grams start with rare characters like 'x' and 'y' or single dashes; 4-grams combine single characters with whitespaces and double dashes; 5-grams cover numbers, in particular, years. Importantly, though, since the best test BLEU scores for IWSLT were obtained by a multi-layer character-based model, the ACT-ENC model has done a reasonable job in improving over the already well-performant strategy, one character per segment, despite having only a single NMT layer. For CASIA and ASPEC, ACT-ENC converged to a segmentation even closer to pure characters: for CASIA, the most frequent 2-grams are punctuation combined with frequent pronoun 他 or preposition 的, a hieroglyph 明 from a common phrase '[smth.] shows, [that]' (all 4-10k in train), and parts of rare English words; for ASPEC, it is mostly the Hiragana letter 'き' that starts the segments. While this letter also occurs as singleton (183× in the dev set, vs. 52× as part of a learned segment), and is frequent in the training set (239k), it is not the most frequent letter. For WMT, charac-

Ref	in social groups of animals , the juveniles always look different than the adults .
Word	in groups of social animals , the children are always different from the other than the <unk>.
BPE	in gruppen sozialer tiere sehen die jung@@ tiere immer anders aus als die alt@@ tiere . in groups , in groups , the juveniles are seeing the same animals as well as the animals .
ACT-ENC	in| g|ru|pp|en| s|oz|ia|le|r |ti|er|e |se|he|n |d|ie| j|un|gt|ie|re| i|m|m|er| a|nd|er|s |au|s |al|s |d|ie| a|lt|ti|er|e |.| in groups , the juvenile seems to see the different approach than the algaes .
Char	in groups of social animals , the juveniles are still in the alite of the altients .
Ref	we 're living in a culture of jet lag , global travel , 24-hour business , shift work .
Word	we live in a civilization with <unk> , global travel , <unk> and <unk> .
BPE	wir leben in einer zivilisation mit jet@@ -@@ lag , weltweiten reisen , non@@ sto@@ p-@@ business und sch@@ icht@@ arbeit . we live in a civilization with a single , a variety of global travel , presidential labor and checking .
ACT-ENC	w|ir| l|eb|en| i|n |ei|ne|r |z|iv|il|is|at|io|n |m|it| j|et|-la|g |,| w|el|tw|ei|te|n |re|is|en| ,| n|on|st|op|-bu|si|ne|ss| u|nd| s|ch|ic|ht|ar|be|it| .| we live in a civilization with jes lag , worldwide rows , nonstop business and failing .
Char	we live in a civilization with jet walk , global journeys , nonstop-business and layering

Table 4: Greedy translations from the IWSLT dev set. Explicit segmentations are given for the ACT-ENC and BPE models.

Data	Len	Segments
IWSLT	2	en; n␣; er; ␣d; ie; e␣; ei; in; ␣s; ␣w …
	3	yst; -␣d; xtr; -␣u; 100; xpe; -␣w; xis; -␣e; -ge …
	4	–␣d; –␣w; –␣s; –␣i; –␣e; –␣u; –␣g; –␣m; –␣a; –␣k …
	5	1965␣; 969␣; 1987␣; 1938␣; 1621␣; 1994␣; 1985␣ …
CASIA	2	”。; ”，; er; ”他; --; ”的; le; 明，; li; ut; …
ASPEC	2	きる; きた; きな; きに; りん; きは; き，; きて …
WMT	2	e␣; s␣; ␣d; t␣; ␣l; es; on; ␣a; de; en …
	3	übe; Rüc; rüb; öve; ürs; Köp; üsl
	4	ümov; ölln; rüng; Jürg; ülle; Müsl Müni; üric; üdig; …

Table 5: Most frequent ACT-ENC segments.

ter 2-grams are all very frequent in the training data (8-11M occurrences) while longer segments are very rare (max. 1k occurrences). Longer segments all include umlauts (ü, ö), which are atypical for French and should be treated as one unit semantically since they are loan words or proper names.

Gating Behavior of Char-GRUs. To investigate the reasons for success of the deep character-based encoders and their better or on-par performance with the segmenting ACT-ENC model, we analyzed average activations of GRU gates. A GRU cell computes the next state as: $s_t = z \odot \tanh(x_t W_h + (h_{t-1} \odot r) W_g) + (1 - z) \odot s_{t-1}$, where z is the update gate and r the reset gate, both being outputs of sigmoid layers receiving x_t and h_{t-1} [25]. Taking a closer look at the average values of these gates, we find patterns of segmentation as depicted in Figure 2 for a 5-layer character model. Most of the time, a whitespace character triggers a visible change of gate behavior: Forward reset gates close (reset) one character after a whitespace and backward reset gates close at whitespaces and then both open at the subsequent character. The update gates show similar regularities, but here the average gate values are less extreme. For longer words all gate activations progressively decay with the length. In addition, the block-wise processing of the compound ‘schreibtisch’ (German: ‘writing table’) that was correctly split into ‘schreib’ and ‘tisch’, points to decompounding abilities that pure character-level models possesses beyond simple whitespace tokenization.

Overall, this illustrates that the recurrent gates equip pure

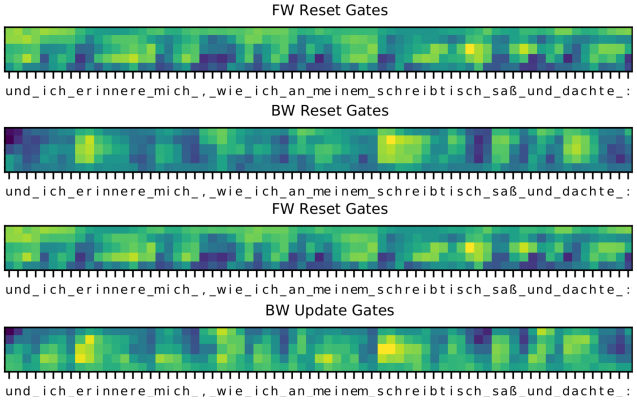

Figure 2: Mean activations for reset and update forward (FW) and backward (BW) GRU gates for an IWSLT sentence as produced by the 5-layer char model. Layers are stacked from bottom to top. Blue: values $\simeq 0$, yellow: values $\simeq 1$.

character models with the capacity to implicitly model input segmentations, which would explain why ACT-ENC could not find a radically different or advantageous segmentation.

5. Summary & Conclusion

We proposed an approach to learning (dynamic and adaptive) input segmentation for NMT based on the Adaptive Computation Time paradigm [9]. Experiments on four translations tasks showed that our model prefers to operate closely to the character level. This is echoed by the quantitative success of pure character-level models (without dynamic segmentation) and a qualitative analysis of gating mechanisms, suggesting that our adaptive model rediscovers the segmenting capacity already present in gated recurrent, pure character-based models. Given this and the absence of many development hurdles with character-based models, their lower memory consumption and higher robustness, the presented dynamic segmentation capacity, being primarily a diagnostic research tool, does not seem to be necessary to be modelled explicitly. We hope these insights can serve as justification for intensification of research in pure character-level NMT models.

6. References

[1] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, *et al.*, "Findings of the 2016 conference on machine translation," in *WMT*, 2016.

[2] ——, "Findings of the 2017 conference on machine translation," in *WMT*, 2017.

[3] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[4] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *ACL*, 2016.

[5] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *ICASSP*, 2012.

[6] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, *et al.*, "Hidden technical debt in machine learning systems," in *NIPS*, 2015.

[7] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *NIPS*, 2017.

[8] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A field guide to dynamical recurrent neural networks*. IEEE Press, 2001.

[9] A. Graves, "Adaptive computation time for recurrent neural networks," in *arXiv:1603.08983*, 2016.

[10] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *ICML*, 2017.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, *et al.*, "Attention is all you need," in *NIPS*, 2017.

[12] J. Chung, K. Cho, and Y. Bengio, "A character-level decoder without explicit segmentation for neural machine translation," in *ACL*, 2016.

[13] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," *ICLR*, 2017.

[14] J. Lee, K. Cho, and T. Hofmann, "Fully character-level neural machine translation without explicit segmentation," *TACL*, vol. 5, pp. 365–378, 2017.

[15] M.-T. Luong and C. D. Manning, "Achieving open vocabulary neural machine translation with hybrid word-character models," in *ACL*, 2016.

[16] S. Zhao and Z. Zhang, "Deep character-level neural machine translation by learning morphology," in *arXiv:1608.04738*, 2016.

[17] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," in *arXiv:1308.3432*, 2013.

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[19] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, *et al.*, "An actor-critic algorithm for sequence prediction," in *ICLR*, 2017.

[20] M. Popovic, "chrF: character n-gram F-score for automatic MT evaluation," in *WMT*, 2015.

[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[22] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, *et al.*, "The best of both worlds: Combining recent advances in neural machine translation," in *arXiv:1804.09849*, 2018.

[23] W. Ling, I. Trancoso, C. Dyer, and A. W. Black, "Character-based neural machine translation," in *arXiV:1511.04586*, 2015.

[24] D. Fojo, V. Campos, and X. Giró-i Nieto, "Comparing fixed and adaptive computation time for recurrent neural networks," in *Workshop Track of ICLR*, 2018.

[25] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, *et al.*, "Learning phrase representations using RNN encoder–decoder for Statistical Machine Translation," in *EMNLP*, 2014.