Leveraging Relaxed Equilibrium by Lazy Transition for Sequence Modeling

Xi Ai
College of Computer Science
Chongqing University
barid.x.ai@gmail.com

Bin Fang
College of Computer Science
Chongqing University
fb@cqu.edu.cn

Abstract

In sequence modeling, certain tokens are usually less ambiguous than others, and representations of these tokens require fewer refinements for disambiguation. However, given the nature of attention-based models like Transformer and UT (universal transformer), all tokens are equally processed towards depth. Inspired by the equilibrium phenomenon, we present a lazy transition, a mechanism to adjust the significance of iterative refinements for each token representation. Our lazy transition is deployed on top of UT to build LT (lazy transformer), where all tokens are processed unequally towards depth. Eventually, LT is encouraged to oscillate around a *relaxed* equilibrium. Our experiments show that LT outperforms baseline models on several tasks of machine translation, pre-training, Learning to Execute, and LAMBADA.

1 Introduction

Attention-based models like Transformer (Vaswani et al., 2017) underly two core concepts: *layer* and *refinement*. With layer-stacked structure, the model entirely relies on attention mechanisms (Parikh et al., 2016; Lin et al., 2017) to refine token representations (e.g.,vectors) layer-by-layer from context-informed representations by using residual connections (He et al., 2016; Ebski et al., 2018). However, $layer$ is not necessary. UT (universal transformer) (Dehghani et al., 2019) and its variants (Bai et al., 2019; Lan et al., 2020) iteratively run single-layer but wide Transformer for sequence modeling that token representations are refined at each step by attending to the sequence representation of the previous step, showing higher performance than layer-stacked Transformer does on NMT (Dehghani et al., 2019), pre-training (Lan et al., 2020), language modeling (Bai et al., 2019), and other tasks of varying complexity (Dehghani et al., 2019) with the same number of parameters. The computational bound of recurrence is not the number of tokens in the sequence (like RNN) or the number of stacked layers in the model but is the maximum number of refinements made to token representations, e.g., a pre-defined maximum number of iteration steps.

We follow this line but further ponder *refinement*. Concretely, the model refines token representations iteratively that tokens are equally processed towards depth for disambiguation. Essentially, different token representations are refined at a step equally and concurrently. Thus, regardless of the ambiguous state, a step shows the same importance for all the tokens. However, certain tokens are less ambiguous than others in sequence modeling, especially in NMT. It raises two questions: *1)* do deep representations have to be learned for less-ambiguous tokens; *2)* do tokens must consider the contributions or importance of a step equally?

Meanwhile, our work derives its motivation from the combination of UT and ACT (adaptive computation time) (Graves, 2016) that can dynamically estimate the importance of iteration steps for understanding different sentences in the bAbI task (Weston et al., 2016). However, we consider the importance of every iteration step for all the tokens in the same sequence, sharing a similar motivation with depth-adaptive Transformer (Elbayad et al., 2020). On the other hand, these methods learn a halting probability for each step without considering the correspondence and interdependence throughout the iteration and the model's convergence. We attempt a mechanism like the reset gate in GRU (Chung et al., 2014) that the model is allowed to dynamically forget some information based on the correspondence and interdependence, where in our case, this information is the further refinement at the current step for the corresponding token representation. In this way, we weaken the importance of an iteration step for less-ambiguous token representations to prevent over-refining but retain the significance for more-ambiguous token

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 2904 - 2924

representations to avoid under-refining, refining token representations at the same step unequally.

To this end, we present a lazy transition that performs between two consecutive steps when running UT iteratively. It forms iterative refinements (Ebski et al., 2018) for token representations with a residual structure:

$$h_i^{t_i} = h_i^{t_i-1} + (1 - \sigma_i^{t_i})UT(h_i^{t_i-1}) \quad (1)$$

where $h_i^{t_i}$ is the representation for the i-th token in the sequence at step t_i (token-wise), UT yields the deep refinement $UT(h_i^{t_i-1})$ for $h_i^{t_i}$ by consuming $h_i^{t_i-1}$, and $(1 - \sigma_i^{t_i})$ is a refining rate for adjusting the contribution of $UT(h_i^{t_i-1})$. In this way, we model an equilibrium (Bai et al., 2019; Lan et al., 2020) between two representations of the same token from consecutive steps. Significantly, when off $(1 - \sigma_i^{t_i}) = 0$, the model is allowed to dismiss the further refinement $UT(h_i^{t_i-1})$ for $h_i^{t_i}$ at the step t_i. Then, the model reaches a local equilibrium for the corresponding token that the input $h_i^{t_i-1}$ is similar to the output $h_i^{t_i}$. In other words, the model ponders the significance of the further refinement via our lazy transition that a token representation is *lazy* to absorb the information if the further refinement is trivial. In sequence modeling, all the equilibrium token representations are learned and concatenated for the required sequence representation that the model oscillates around a *relaxed* equilibrium.

Our contributions are: *1)* we present a lazy transition that is placed on top of UT to build LT (lazy transformer). Our lazy transition dynamically forms a refinement path for a token at each step by pondering the step importance and the local equilibrium. Eventually, LT is encouraged to oscillate around a *relaxed* equilibrium in sequence modeling; *2)* we provide an empirical study to quantitatively analyze how the *relaxed* equilibrium impacts NMT in performance, deep models, and zero-shot inferring; *3)* we show our model can consistently improve the performance of pre-training and two tasks of varying complexity, where standard Transformer fails; *4)* our empirical study shows that stable and smooth refinements at the early iteration steps are significant, which results in a strong equilibrium and a stable oscillation.

2 Related Work

Iterative Refinement in Transformer, Tied Transformer, and Universal Transformer One core backbend of Transformer (Vaswani et al., 2017) is iterative refinements (Ebski et al., 2018) that are finished by forming a residual structure around each sub-layer: $h_i^l = h_i^{l-1} + f^l(h_i^{l-1})$, where l denotes depth and $f(\cdot)$ represents a sub-layer (e.g., an attention layer (Parikh et al., 2016; Lin et al., 2017)). Following tied Transformer (Gulcehre et al., 2018; Xia et al., 2019; Dabre and Fujita, 2019), which share parameters across some layers, (Dehghani et al., 2019) introduce token-wise recurrence (Graves et al., 2014; Joulin and Mikolov, 2015; Kaiser and Sutskever, 2016) to signal-layer but wide Transformer and then present UT, a Turing-complete model. Theoretically, UT yields a similar recurrent inductive bias of RNN's because each token representation is refined by attending to the sequence representation of the previous step with tied parameters. We present a comparison in Appendix A. Significantly, the iterative refinement in UT is reformed to : $h_i^t = h_i^{t-1} + f(h_i^{t-1})$, where f is the same for every iteration step t, i.e., a single-layer structure that the parameters are tied throughout the iteration. In this work, we apply our lazy transition on top of a UT block. To reform iterative refinements, we are inspired by (Zhang et al., 2021; Escolano et al., 2021; Bapna et al., 2018; Wang et al., 2019) that f (or f^l) could be a more complicated network rather than a simple sub-layer, defining f as the entire UT block and reforming the iterative refinement as Eq.1.

Adaptive Computation While a token-wise recurrent model like UT can theoretically run infinitely, the model is commonly trained by setting a maximum number of iteration steps. However, the number of inference steps is flexible. (Graves, 2016) first introduce ACT (Adaptive Computation Time) to compute a scalar halting probability predicted by the model at each step for each token in RNN. Then, (Dehghani et al., 2019) adapt ACT for UT to halt the iteration before reaching the maximum number of training steps[1] in inferring. Furthermore, (Elbayad et al., 2020) present depth-adaptive decoding for tokens to model the distribution of exiting with the probability of computing each layer/step and then emitting token predictions. Our lazy transition follows this line somewhat that $(1 - \sigma_i^{t_i})$ in Eq.1 dynamically approximates the computation time for tokens. It reflects on two characteristics: *1)* $(1 - \sigma_i^{t_i}) = 0 \equiv$ stop iteration;

[1]In some papers, training steps refer to how many batches we use for training. In this paper, training steps mean "how many steps we iterate a model for a training batch."

2) $(1-\sigma_i^{t_i})$ computed at each step.

3 Approach

We present LT (Lazy Transformer), a token-wise recurrent model. LT only consists of a UT block and our lazy transition. In training, we iteratively run LT but constrain the computational bound to control the training process by setting a maximum number of iteration steps. However, iteration steps are dynamic in inferring.

3.1 Relaxed Equilibrium

In sequence modeling, DEQ(Bai et al., 2019) and ALBERT (Lan et al., 2020) report that a UT-based model tends to oscillate around an equilibrium. Concretely, each additional step has a smaller and smaller contribution to the current sequence representation until the model oscillates around a fixed point. Formally, we have: $\lim_{t\to+\infty} h^t_{1:N} = \lim_{t\to+\infty} UT(h^t_{1:N}) \equiv h^E_{1:N} = UT(h^E_{1:N})$, where t is the step, $h^t_{1:N}$ is the sequence representation of a N length sequence $X_{1:N}$ at t, and $h^E_{1:N}$ is the equilibrium sequence representation. Empirically, the equilibrium phenomenon could be observed from the difference norm of sequence representations: $||UT(h^t_{1:N}) - h^t_{1:N}|| \approx ||UT(h^{t+1}_{1:N}) - UT(h^t_{1:N})|| \equiv ||UT(h^{t:}_{1:N}) - h^{t:}_{1:N}|| < \epsilon$, where ϵ depends on the model and $t:$ denotes steps after t.

Intuitively, UT, DEQ, and ALBERT are encouraged to reach a global and sequence-level equilibrium[2] at a specific iteration step t in a dynamic programming style for outputting the equilibrium sequence representation. By contrast, we attempt a greedy strategy to independently find a local equilibrium for a token, modeling the locally optimal choice. Then, for an input sequence, we model all the local equilibriums to find a *relaxed* equilibrium instead of naively finding a global and sequence-level equilibrium. Formally, in sequence modeling, we model a *relaxed* equilibrium to obtain an equilibrium sequence representation $h^R_{1:N}$:

$$\begin{aligned}
&\forall i \in N : \lim_{t_i\to+\infty} h^{t_i}_i = \lim_{t_i\to+\infty} UT(h^{t_i}_i) \\
&\equiv \forall i \in N : h^{R_i}_i = UT(h^{R_i}_i) \\
&\equiv h^R_{1:N} = UT(h^R_{1:N}) = [h^{R_1}_1, h^{R_2}_2, ..., h^{R_N}_N]
\end{aligned} \tag{2}$$

where $h^{t_i}_i$ denotes i-th token representation at t_i (token-wise), $h^{R_i}_i$ is the equilibrium token representation, R stands the step of reaching the *relaxed* equilibrium, and $[\cdot]$ denotes concatenation. Similarly, we can evaluate the local equilibrium from the difference norm of token representations:

$$||UT(h^{t_i:}_i) - h^{t_i:}_i|| < \tilde{\epsilon} \tag{3}$$

Note that, in our preliminary experiment, we find that $h^E_{1:N} \neq [h^{R_1}_1, h^{R_2}_2, ..., h^{R_N}_N]$, i.e., the equilibrium sequence representation is not equivalent to the combination of all the equilibrium token representations because tokens have different ambiguous states. For the *relaxed* equilibrium, we do not have to update tokens equally at an iteration step so that an iteration step can show varying importance for different tokens, subject to the ambiguous state of the token representations. Thus, we consider adjusting the impact of the current step for different tokens throughout the iteration.

3.2 Equilibrium from Linear CKA

Although we can easily and immediately observe the equilibrium or the local equilibrium phenomenon from difference norms, we have no prior knowledge in practice that we cannot analyze the equilibrium quantitatively throughout the iteration. Hence, we consider the linear CKA exam (centered kernel alignment): $CKA(X,Y) = ||X^TY||^2_F / (||X^TX||_F ||Y^TY||_F) \in (0,1]$ that is introduced by (Kornblith et al., 2019) to identify correspondences between representations in models trained from different initializations and is invariant to orthogonal transform and isotropic scaling but is not invariant to any linear transforms. For this exam, we are inspired by (Wu et al., 2020), who measure the degree of a layer's multilinguality with a CKA exam between two averages of outputted sequence representations. However, we run the exam for two averages of sequence representations $h_{1:N}$ emerged from two consecutive steps[3]. Precisely, we majorly rewrite the equilibrium to:

$$\begin{aligned}
&\lim_{t\to+\infty} CKA(\tilde{h}^t, \frac{\sum_{i=1}^N UT(h^t_i)}{N}) = 1 \\
&\equiv CKA(\tilde{h}^E, \frac{\sum_{i=1}^N UT(h^E_i)}{N}) = 1
\end{aligned} \tag{4}$$

[2] Our preliminary experiment confirms this intuition, which will be further discussed in §Experiment.

[3] Note that, in this way, the feature space is the channel of representations, not the representations itself in the original CKA exam.

where $\tilde{h}^t = \frac{\sum_{i=1}^{N} h_i^t}{N}$. For our *relaxed* equilibrium in sequence modeling, we have:

$$\begin{aligned} &\forall i \in N : \lim_{t_i \to +\infty} CKA(h_i^{t_i}, UT(h_i^{t_i})) = 1 \\ &\equiv \forall i \in N : CKA(h_i^{R_i}, UT(h_i^{R_i})) = 1 \\ &\equiv \sum_{i=1}^{N} CKA(h_i^{R_i}, UT(h_i^{R_i})) = N \end{aligned} \tag{5}$$

Essentially, we can dynamically evaluate the local equilibrium by giving the exam to token representations throughout the iteration:

$$CKA(h_i^{t_i:}, UT(h_i^{t_i:})) \approx 1 \tag{6}$$

It reflects the correspondence and interdependence between UT's input and output. Note that, although the global equilibrium does not expect the local equilibrium, this exam also gives an intuition of how a token representation changes throughout the iteration.

3.3 Lazy Transition

To leverage the *relaxed* equilibrium (Eq.2 and Eq.5), our lazy transition uses a residual structure to form iterative refinements (Ebski et al., 2018) for h_i at each step in order to obtain $h_i^{R_i}$. Recall that, $UT(h_i^{t_i-1})$ is the deep refinement we can obtain at step t_i, and $CKA(\cdot)$ returns $(0, 1]$. We form the iterative refinement (Eq.1) to:

$$\begin{aligned} h_i^{t_i} &= h_i^{t_i-1} + (1 - \sigma_i^{t_i}) UT(h_i^{t_i-1}) \\ \sigma_i^{t_i} &= CKA(h_i^{t_i-1}, UT(h_i^{t_i-1})) \end{aligned} \tag{7}$$

Concretely, the model is based on the correspondence and interdependence between UT's input $h_i^{t_i-1}$ and output $UT(h_i^{t_i-1})$. When $\sigma_i^{t_i}$ is close to 1 at step t_i, $h_i^{t_i:}$ is only oscillating around $h_i^{t_i}$, and UT cannot provide useful information for better representations anymore. Then, the model is encouraged to dismiss UT and then outputs h_i's equilibrium token representation: $h_i^{R_i} (\equiv h_i^{t_i-1} \equiv h_i^{t_i} \equiv h_i^{t_i:})$, for the the *relaxed* equilibrium. It is similar to the reset gate in GRU that dynamically forget the previous state, but our lazy transition attempts to dismiss the newly obtained information $UT(h_i^{t_i-1})$ that is unimportant. Therefore, our lazy transition provides implicit step information, similar to GRU that can identify the current position and handle input sentences of varying length. Since there is no variable for identifying steps, the model can run varying steps in inferring. On the other hand, this could be viewed as a linear interpolation: $h_i^{t_i} = \sigma_i^{t_i} h_i^{t_i-1} + (1 - \sigma_i^{t_i})(h_i^{t_i-1} + UT(h_i^{t_i-1}))$, where $\sigma_i^{t_i}$ decides how much the model updates $h_i^{t_i-1}$ at step t_i from the pre-refined representation[4]: $h_i^{t_i-1} + UT(h_i^{t_i-1})$.

Meanwhile, since $h_i^{t_i}$ is informed by $UT(h_i^{t_i-1})$ without any step-specific parameter, the benefits are twofold: *1)* our lazy transition does not hurt the global receptive field of UT, only adjusting the contribution of a step; *2)* the recurrent inductive bias of UT is inherited because all the parameters are tied throughout the iteration. Our empirical study confirms these two benefits (see §Experiment).

3.4 Lazy Transformer

In sequence modeling, since our model does not provide any position information, in order for the model to make use of the order of the sequence, we inject position information at each step. Therefore, our LT (lazy Transformer) is formed as: $h_{1:N}^{t_{1:N}} = h_{1:N}^{t_{1:N}-1} + (1 - \sigma_{1:N}^{t_{1:N}}) UT(h_{1:N}^{t_{1:N}-1} + PE_{1:N})$, where $h_i^{t_i}$ computed by Eq.7 is the i-th token representation of a N length sequence representation $h_{1:N}$ and $PE_{1:N}$ is the sinusoidal position encoding for identifying positions as defined in (Vaswani et al., 2017). Recall that $1 - \sigma_i^{t_i}$ collapses to 0 when the model is researching the local equilibrium of h_i. The model can simply copy the equilibrium token representation to the next step for speed.

Instantiation LT can be instantiated, used, and trained as the same as a vanilla Transformer block or a UT block. Concretely, we can instantiate and train: *1)* a LT encoder with the objective of MLM (masked language modeling) (Devlin et al., 2019; Lan et al., 2020); *2)* a LT decoder with the objective of GPT (generative pre-training) (Radford et al., 2018; Alec Radford, 2020); *3)* a LT encoder-decoder (consisting of a LT encoder and a LT decoder) in a *seq2seq* (Graves, 2013) manner.

3.5 Comparison

Readers can refer to Appendix C for details.

Lazy Transition vs. GRU They have different motivations. GRU aims to learn a segment-level representation by accumulating the information from all the tokens, whereas LT is encouraged to ponder the importance of a step for different tokens and then to oscillate around the *relaxed* equilibrium.

[4] In other words, we assume $\sigma_i^{t_i} = 0$ that all the newly computed information $UT(h_i^{t_i-1})$should be added to $h_i^{t_i-1}$.

Meanwhile, compared to GRU, which forgets the previously computed state via the reset gate, our lazy transition is allowed to dismiss the newly computed information that is trivial.

Lazy Transition vs. Adaptive Computation Adaptive computation methods like ACT (Graves, 2016) and Adaptive-depth Transformer (Elbayad et al., 2020) learn a generator to output a probability of exiting based on the step output. We argue that these methods are agonistic for the model's convergence because they do not consider the information flow from the input to the corresponding output. By contrast, our method leverages the model's convergence and applies CKA to ponder the correspondence and interdependence between inputs and outputs, where in our case, the model's convergence is the *relaxed* equilibrium.

Lazy Transformer vs. UT, DEQ, and ALBERT LT is parallel to UT (Dehghani et al., 2019), DEQ (Bai et al., 2019), and ALBERT (Lan et al., 2020). We share the idea of recurrence over depth, but we have three main differences: *1)* previous methods require an explicit step encoding for each iteration, whereas we let our lazy transition handle iterations implicitly; *2)* we ponder the significance of a step for different tokens, whereas previous methods refine tokens equally at a step; *3)* we consider the local and token-level equilibrium in addition to the sequence-level equilibrium.

4 Experiment

We divide our empirical studies and experiments into two genres: *1)* we experiment with NMT (our main task) to confirm the effectiveness of our methods for large-scale sequence modeling and further quantitatively justify our hypotheses and assumptions; *2)* we attempt pre-training tasks and two somewhat rare but challenging tasks: the Learning to Execute task (Zaremba and Sutskever, 2014) and the LAMBADA (Paperno et al., 2016) task, to observe the performance on tasks of varying complexity. All the links of datasets, libraries, scripts, and tools marked with ⋄ are listed in Appendix H. We open source code on GitHub.

Training Our code is implemented on Tensorflow 2.6 (Abadi et al., 2016) with 4 NVIDIA TITAN Xp 12G GPU. We implement our model based on the codebase of official UT from tensor2tensor⋄[5] and official CKA⋄. We use the default setting: *universal_transformer_base* from tensor2tensor. Concretely, we use Adam optimizer (Kingma and Ba, 2015) with parameters $\beta_1 = 0.9$,$\beta_2 = 0.997$ and $\epsilon = 10^{-9}$, and a dynamic learning rate with $warm\_up = 8000$ (Vaswani et al., 2017) ($learning\_rate \in (0, 7e^{-4}]$) is employed. We set dropout regularization with a drop rate $rate = 0.1$ and label smoothing with $gamma = 0.1$ (Mezzini, 2018). For data feeding efficiency, each batch of similar-length sequences are padded to the same length and may have a different number of elements in each batch.

Reimplementation and Reconfiguration We reimplement some models on our machine with the same batch size. We compare the reimplemented results to the reported results on the same test set to ensure the difference is less than 5% (or 1 in BLEU). Then, we can confirm the reimplementation and reconfiguration.

4.1 Neural Machine Translation

Dataset and Preprocessing We train a LT encoder-decoder model for machine translation. To be comparable, we share two NMT tasks: *1)* $English \rightarrow German$ of WMT 2014⋄ (Bojar et al., 2014); *2)* $English \rightarrow Romanian$ of WMT 2016⋄ (Bojar et al., 2016). Following the standard evaluation, the model is evaluated on *newstest2014* for $English \rightarrow German$ and *newstest2016* for $English \rightarrow Romanian$. We use the Moses tokenizer⋄ developed by (Koehn et al., 2007) for tokenization and use fastBPE⋄ to learn shared 32K BPE (Sennrich et al., 2016) for a language pair. Data filtering is finished by FAIR tool⋄ (Ng et al., 2019). We use *sacreBleu*⋄ (Post, 2018) with standard settings[6] to evaluate the quality of translation.

Model Configuration The model configurations are identical to base-UT (Dehghani et al., 2019). Specifically, we set model dimension, word embedding, head, and FFN filter to 1024, 1024, 16, and 4096, which results in the same number of parameters (62M) as base-Transformer

[5]Note that, the newest UT implementation uses pre-normalization $y = x + f(ln(x))$ instead of reported post-normalization $y = ln(x + f(x))$. UT with pre-normalization shows slight degradation in performance ($\approx 1\%$) but improves stability in training, initialization, and scalability, where x denotes the input, $f(.)$ stands for a sub-layer, and ln is a layer-normalization unit.

[6]{nrefs:1|case:mixed|eff:no|tok:13a|smooth:exp|version:2.0.0}

#	Model	newstest2014 $En \rightarrow De$	newstest2016 $En \rightarrow Ro$
	base model: $\{6,6\} \implies \{6,6\}$		
1	base-Transformer (Vaswani et al., 2017)	27.50	32.31
2	⋆base-UT w/o SE	27.85	
3	⋆base-UT	28.73	33.97
4	base-UT (Dehghani et al., 2019)	28.90	
5	⋆base-UT + ACT	29.11	
6	⋆base-UT + GRU	26.59	
7	OURS: base-LT	29.81	35.02
	deep model: $\{20/40,6\} \implies \{20/40,6\}$		
9	⋆ 20-Transformer (Bapna et al., 2018)	28.72	33.59
10	20-Transformer (Wang et al., 2019)	28.90	
11	⋆20-UT	29.69	34.32
12	OURS: 20-LT	30.54	35.62
13	OURS: 40-LT	31.05	36.04

Table 1: Performance of translation. ⋆ *denotes the baseline models that are reimplemented.*. SE denotes step encodings.

(Vaswani et al., 2017). Beam search is configured with beam size 4 and length penalty 0.6. We train the model for 100k iterations. We use $\{T\_enc\_step, T\_dec\_step\} \implies \{I\_enc\_step, I\_dec\_step\}$ to denote a model that runs the maximum T_enc_step and T_dec_step steps in the encoder and decoder respectively for training and runs the maximum I_enc_step and I_dec_step steps in the encoder and decoder respectively for inferring. For instance, $\{6,6\} \implies \{6,6\}$ means we set the maximum step to 6 both in the encoder and decoder for training and inferring.

4.1.1 Base Model

In this experiment, we set $\{6,6\} \implies \{6,6\}$ (**base-LT**), which is identical to the baseline model: base-UT. Also, it is equivalent to base-Transformer that has 6 layers in both the encoder and decoder. For comparison, we place GRU [7] on top of the UT block for evaluation (base-UT + GRU), similar to that we use our lazy transition, and we also train another UT with ACT (base-UT + ACT). All of these models *require* step encodings for the identification of steps. For the evaluation of how our lazy transition handles iterations without explicit encodings, we instantiate another UT without step encodings (base-UT w/o SE), a similar model that naively repeats one layer without step identifications (Dabre and Fujita, 2019).

Table 1 shows the results on NMT tasks. base-LT outperforms base-UT (row 3&7) by 4%. By observing row 2&3, the performance of UT significantly degrades without using step encodings, which indicates a mechanism for step identification is beneficial for UT. Meanwhile, we find that applying ACT to UT (row 5) can improve the performance on NMT, but GRU (row 6) seems to have no effect.

We observe the equilibrium phenomena in models by giving the CKA exam (Eq.6) in the encoder and decoder. Visualizations are presented in Appendix E. In our case study, all the tokens in base-UT (Appendix E.1 (c,d)) run to an equilibrium synchronously because every step shows similar importance to all the tokens, i.e., all the tokens have similar CKA scores at every step throughout the iteration. It confirms that base-UT tends to find a global equilibrium (Eq.4) for all the tokens, as discussed before. Furthermore, this process is unstable that CKA scores change dramatically at the early steps[8], resulting in unstable refinements (under-refining or over-refining) for some tokens and hurting tokens' local equilibrium. Meanwhile, we are aware that base-UT + ACT (Appendix E.3), base-UT w/o SE (Appendix E.2), and base-UT + GRU (Appendix E.4) show a similar behavior to base-UT throughout the iteration. base-UT + GRU is even more irregular than base-UT at the early steps. Also, we suspect that ACT and GRU have no power to adjust the importance of a step for each token throughout the iteration in NMT.

Compared to that, base-LT (Appendix E.1 (a,b)) enables each token to smoothly find its local equilibrium independently and then turns to oscillate around the *relaxed* equilibrium (Eq.2) together. Specifically, we observe that CKA scores of a step ($\sigma_i^{t_i}$ in Eq.7) differ from one token to the others at every step in the early 4 iterations, which answers our question in §Introduction that a step can show varying importance for all the tokens. Then, base-LT tends to refine token representations based on the sequence-level characteristics at the late 2 steps to oscillate around the *relaxed* equilibrium, resulting in similar CKA scores.

4.1.2 Deep model

Given the nature of the iteration-based model, we can run infinite steps in the encoder and decoder. Therefore, we test the performance beyond 6 steps. In this experiment, we share a challenge with (Bapna et al., 2018; Wang et al., 2019) to train deep models. According to their works, NMT can get significant benefits from many layers in the encoder but not in the decoder. Similarly, we assume we can obtain benefits from running many steps in the encoder. For comparison, we config-

[7] We use orthogonal kernels for GRU to solve an optimization problem. See Appendix B for details.

[8] e.g., from step 1 to step 2, the average of absolute difference is 0.221, which is only 0.105 in base-LT.

ure models with $\{20, 6\} \implies \{20, 6\}$ (**20-LT**), which is similar to the 20-layer model in (Bapna et al., 2018; Wang et al., 2019). Beyond 20 steps in the encoder, we further configure a model with $\{40, 6\} \implies \{40, 6\}$ (**40-LT**).

We show the results of deep models in Table 1. Running a large number of steps in the encoder can consistently improve the performance of LT, and deep LT outperforms baseline models. Similar to base-LT, we also consider the CKA exam for the equilibrium phenomena (Appendix E.5) that we obtain similar conclusions to base-LT. It indicates our method is a potential new development for deep models without introducing additional parameters.

Besides, we find UT can get benefits from deep settings, but the performance is not very strong. Intuitively, it is caused by the global equilibrium strategy observed in base-UT that CKA scores change dramatically at the early steps. Therefore, 20-UT cannot smoothly run to an equilibrium, which results in a suboptimal choice. The CKA exam confirms our intuition and draws a similar conclusion of base-UT. By contrast, instead of searching a global equilibrium from scratch, LT searches local equilibriums for tokens at the early steps, which results in varying contributions for different token representations, and then turns to find a global equilibrium for all tokens that we observe a similar behavior to UT. We conjure that LT tends to focus on token-specific characteristics first and then on sequence-specific characteristics.

4.1.3 Zero-shot Inferring

In the above experiments, we set the inference steps as the same as the training steps, sharing the same strategy with previous works (Dehghani et al., 2019; Lan et al., 2020; Bai et al., 2019). We are interested in an asymmetric strategy that the inference steps is different from the training steps. Thus, some steps are zero-shot inferring without explicit training. Recall that the model oscillates around an equilibrium point and outputs similar representations (if iterating). We assume these similar representations can be used for prediction, and we expect similar performance. On the other hand, we visualize the equilibrium phenomena to observe the equilibrium, and zero-shot inferring can quantitatively examine the equilibrium. For this test, we reuse our trained models[9]

Encoders / Decoder	4	5	6	7	8	20
base-UT:$\{6, 6\} \implies \{X, X\}$ (nonzero: $avg : 22.16$, $std : 6.32$)						
4	24.17	27.67	24.63	24.73	23.16	13.00
5	26.55	27.32	25.49	25.84	26.41	17.89
6	26.47	24.66	**28.73**	25.92	23.97	23.74
7	25.24	28.89	28.95	28.58	25.34	22.79
8	24.04	27.70	28.95	27.00	24.04	23.67
12	21.25	29.57†	29.22	26.96	27.7	14.21
20	14.95	21.4	21.6	22.48	22.33	14.21
base-LT:$\{6, 6\} \implies \{X, X\}$ (nonzero: $avg : 24.52$, $std : 5.03$)						
4	24.28	26.23	26.88	24.13	23.61	27.77
5	21.35	25.41	25.93	26.16	24.96	25.63
6	24.46	26.35	**29.81**	27.45	24.41	24.30
7	24.83	29.45	26.61	27.27	27.08	27.51
8	24.74	28.70	27.86	27.03	27.48	24.51
11	26.13	31.47†	28.35	29.35	29.61	25.16
20	23.87	28.11	28.07	28.15	26.5	25.80
20-UT:$\{20, 6\} \implies \{X, X\}$ (nonzero: $avg : 26.03$, $std : 4.62$)						
18	24.02	26.28	26.82	27.48	29.11	15.73
19	24.02	26.24	27.02	27.48	29.18	13.27
20	23.97	26.2	**29.69**	27.48	27.45	16.72
21	23.55	26.14	27.23	26.92	28.23	14.29
22	22.93	25.88	27.27	26.76	28.29	15.76
26	22.13	26.86	27.19	30.08	30.96†	11.54
40	21.42	24.70	27.15	27.37	25.55	10.05
20-LT:$\{20, 6\} \implies \{X, X\}$ (nonzero: $avg : 28.89$, $std : 1.65$)						
18	26.17	27.26	26.82	27.82	29.82	32.05†
19	26.17	27.26	26.85	27.82	29.82	30.80
20	26.17	26.98	**30.54**	27.65	29.82	30.80
21	25.78	26.90	27.63	27.90	29.45	30.29
22	26.23	26.90	27.63	27.90	29.45	30.29
40	25.29	26.86	27.25	27.84	28.52	30.89
40-LT:$\{40, 6\} \implies \{X, X\}$ (nonzero: $avg : 29.31$, $std : 1.54$)						
39	26.24	27.92	30.05	28.35	29.65	31.59
40	26.24	28.82	**31.05**	29.29	30.18	31.59
41	25.55	28.82	31.05	28.77	30.18	31.59
42	25.55	28.82	31.05	28.77	31.11	31.59
44	25.55	28.82	31.05	28.77	31.11	31.59†
50	25.46	28.29	31.05	28.27	30.64	31.48
base-UT w/o SE:$\{6, 6\} \implies \{X, X\}$ (nonzero: $avg : 21.04$, $std : 7.95$)						
base-UT + GRU:$\{6, 6\} \implies \{X, X\}$ (nonzero: $avg : 17.45$, $std : 9.13$)						
base-UT + ACT:$\{6, 6\} \implies \{X, X\}$ (nonzero: $avg : 23.81$, $std : 6.75$)						

Table 2: Partial results of zero-shot inferring. Training steps are in bold. SE denotes step encodings. † denotes the best performance. See texts for description.

for $En \rightarrow De$ but change the setting to $\{6, 6\} \implies \{[1, 12], [1, 12]\}$, $\{20, 6\} \implies \{[15, 26], [1, 12]\}$, and $\{40, 6\} \implies \{[35, 46], [1, 12]\}$ respectively. For metrics, we compute avg and std for $nonzero$ (> 1) outputs, where avg indicates how strong the equilibrium point is and std tells us how far the model is oscillating around the point.

We show a part of experimental results in Table 2 (see all the results in Appendix F). We observe that models can achieve competitive performance from zero-shot inferring for some steps close to the training step, and the best performance does not precisely exist at the training step, except for base-UT + ACT. Essentially, we observe and confirm some conclusions as mentioned earlier. **1)** LT's behavior is regularized around the equilibrium when comparing base-LT std : 5.03 with base-UT std : 6.32 and 20-LT std : 1.65 with 20-UT std : 4.62. Specifically, LT's lazy transition controls the refinements throughout the iteration that token representations from one step can jointly compose a sequence representation yielding similar performance to the others, whereas UT is

[9]Interestingly, we find the reported sinusoidal step encodings for UT support this test without bringing noises to untrained inference steps. However, some absolute methods or look-up methods are problematic.

encouraged to compose the final sequence representations at the training step and lacks a mechanism to control the refinements explicitly. **2)** UT does not oscillate around a strong equilibrium point, i.e., having a suboptimal equilibrium, because base-UT avg : 22.16 and 20-UT avg : 26.03 are not strong. By contrast, LT is stably oscillating around a strong equilibrium and can compose relatively strong sequence representations in different inference steps because base-LT and 20-LT achieve avg : 24.52 and avg : 28.89 respectively. **3)** In both UT and LT, deep models generally find a stronger and more stable equilibrium than base models. **4)** base-UT + GRU fails in this test, which results in low avg : 17.46 and high std : 9.13. We suspect this failure causes by a very irregular behavior at the first step, as the CKA exam indicates in Appendix E.4. **5)** base-UT + ACT outperforms base-UT on avg. However, base-UT is slightly more stable than base-UT + ACT. Besides, base-UT + ACT generally has a larger gradient norm than others, which may impact the stability and convergence in training. *6)* Due to the halting mechanism in ACT, base-UT + ACT seems to halt the process instead of oscillating around the equilibrium because the performance is constant after some steps. **7)** base-UT significantly outperforms base-UT w/o SE on stability, which indicates a mechanism for step identifications is an essential component for stability.

Limitation We do not find a solution to pick the inference step for the best performance, because we do not recognize any pattern. Meanwhile, we could dynamically select a step for a tradeoff between speed and performance by simply copying equilibrium token representations from the previous steps to the next steps. We will leave this for our future work. Besides, we find LT and UT are not stable when translating sentences longer than 50 words. The BLEU score varies from 5 to 60 for different sentences (see Appendix G). We will conduct further experiments to probe this problem.

4.2 Pre-training Task

ALBERT (Lan et al., 2020) study the application of UT in pre-training. Since LT is extended from UT, we study LT in pre-training, sharing the framework of ALBERT. Concretely, our setting is identical to base ALBERT, denoting it as 12-base-ALBERT. We set the model dimension, word embedding dimension, and the maximum number of steps to 768, 128, and 12. Note that in the original ALBERT, they denote steps as layers. As recommended, we generate masked span for the MLM targets using the random strategy from (Joshi et al., 2020), and we use LAMB optimizer$\diamond$ with a learning rate of 0.00176 (You et al., 2020) instead of Adam optimizer. The only change is that we use our LT to replace UT in 12-base-ALBERT, and we denote our model as 12-base-LT. Following the instructions, we pre-train models on BooksCorpus$\diamond$ (Zhu et al., 2015) and English Wikipedia$\diamond$ (Devlin et al., 2019) for 140k steps. Then, we fine-tune on MNLI$\diamond$ (Williams et al., 2018) and SQuAD(v1.1 and v2.0)$\diamond$ (Rajpurkar et al., 2016, 2018). We report the performance on the dev set as the same as (Devlin et al., 2019; Lan et al., 2020).

Model	SQuAD1.1 (F1)	SQuAD2.0 (F1)	MNLI (Acc)
12-base-ALBERT (Lan et al., 2020)	89.3	80.0	81.6
⋆12-base-ALBERT	89.4	80.0	81.4
⋆12-base-ALBERT-ACT	89.5	80.5	81.6
⋆12-base-ALBERT-GRU	86.9	77.8	78.6
OURS: 12-base-LT	89.8	81.1	82.1
⋆24S-base-ALBERT	89.6	80.9	81.7
OURS: 24-base-LT	90.1	81.7	82.6

Table 3: LT in pre-training. ⋆ *denotes the baseline models that are reimplemented.*

Result We report the result in Table 3. In this test, we run all models 12 steps, and we implement 12-base-ALBERT-ACT and 12-base-ALBERT-GRU, similar to the experiment of the translation task. 12-base-LT significantly outperforms 12-base-ALBERT, 12-base-ALBERT-ACT, and 12-base-ALBERT-GRU. These observations confirm the effectiveness of our lazy transition. For further tests, we train all models in 24 steps. Our model gets benefits from a large number of steps, improving the performance from the base model significantly. By contrast, 24-base-ALBERT cannot obtain significant improvements.

4.3 Learning to Execute

LTE (Learning to Execute) (Zaremba and Sutskever, 2014) including program evaluation tasks (program, control, and addition) and memorization tasks (copy, double, and reverse) is an algorithmic task of varying complexity. The goal is to train models on short snippets of python code to predict the output of the generated programs, which is parameterized by their *length* and *nesting*. Specifically, *length* is the number of digits in the integers that appear in the programs, and *nesting*

	program evaluation			memorization		
Model	program	control	addition	copy	double	reverse
⋆LSTM	54.1	69.2	83.8	78.1	51.9	92.1
⋆Transformer (Vaswani et al., 2017)	72.0	92.9	99.8	98.2	94.8	81.8
DNC (Graves et al., 2016)	69.5	83.8	99.4	100.0	100.0	100.0
Entet (Henaff et al., 2017)	73.4	83.8	98.4	91.8	62.3	100.0
RMC (Santoro et al., 2018)	79.0	99.6	99.9	100.0	99.8	100.
UT (Dehghani et al., 2019)	89.0	100.0	100.0	100.0	100.0	100.
OURS: LT	91.2	99.8	100.0	100.0	100.0	100.

Table 4: Test per character Accuracy on LTE. ⋆ *denotes the baseline models that are reimplemented.*

is the number of times we are allowed to combine the operations. Following the instructions from the official repository of LTE◇, we use the mix-strategy to generate the datasets for training.

Result Table 4 shows the results on LTE. Our method yields benchmark performance on the program task of program evaluation (column 2) and reaches SOTA performance on the other tasks.

4.4 LAMBADA Language Modeling

We attempt the LAMBADA (Paperno et al., 2016) task to evaluate our model on language modeling tasks of varying complexity. The goal of the LAMBADA task is to predict the target word of the target sentence, based on a narrative passage. In this test, we only use the setting of the standard setup as language modeling that is more challenging. Following the instructions (Parikh et al., 2016), we download the dataset from the official repository of LAMBADA◇, and then we train the model to predict the next word as a general language modeling task on the training dataset but only predict the target word at test time. *Note that we do not compare our method with pre-training-based SOTA* (Radford et al., 2018; Brown et al., 2020; Alec Radford, 2020). Readers can refer to Appendix D or the authors' paper for more introduction.

Result Table 5 shows the results on the LAMBADA task. **1)** We first observe that Transformer fails in this test. Specifically, Transformer shows strong performance on *control* but weak performance on *test*. The low performance on *test* cannot be attributed simply to poor language modeling because *control* is used to evaluate Transformer in standard language modeling before *test*. We suspect that the low performance on *test* can be attributed to a lack of inductive bias in training. Concretely, Transformer is trained to predict the next word as a general language modeling task but only predict the target word at test time. The varying complexity leads to failure on *test*, similar to the report in (Dehghani et al., 2019). **2)** Our method significantly improves the performance on *test*, which means our method does not hurt the recurrent inductive bias inherited from UT [10]. Also, our method is robust to the maximum number of steps we choose (base-LT achieves strong results), whereas UT seems a bit sensitive to the maximum number of steps.

	Ppl. (Acc. %)		
Model	dev	control	test
N-GRAM (Paperno et al., 2016)	3125 (0.1)	285 (19.1)	
N-GRAM+Cache (Paperno et al., 2016)	768 (0.1)	270 (19)	
⋆Transformer	5331 (0)	141 (18)	7433 (0.0)
⋆ LSTM	5211 (0)	139 (22)	5314 (0.0)
Neural Cache Model (Grave et al., 2017)	129	139	
base-UT(Dehghani et al., 2019)	279 (18)	131 (32)	319 (17)
ACT+base-UT(Dehghani et al., 2019)	134 (22)	130 (32)	142 (19)
8-UT(Dehghani et al., 2019)	192 (21)	129 (32)	202 (18)
OURS: base-LT	122 (23)	124 (32)	128 (20)
OURS: 8-LT	114 (24)	110 (33)	119 (21)

Table 5: LAMBADA challenge. ⋆ *denotes the baseline models that are reimplemented.* We show language modeling perplexity (Ppl., lower better) with accuracy (Acc., higher better) in parentheses.

5 Conclusion

In this work, we place our lazy transition on top of UT to build LT. Our lazy transition leverages the *relaxed* equilibrium for sequence modeling and provides step identifications that the model ponders the importance of every step for different tokens throughout the iteration. Our main experiment shows that LT can achieve strong performance on translation tasks, facilitate the training of deep models, and tackle the challenge of zero-shot inferring. Our method retains the recurrent inductive bias learned by its UT component, which is confirmed by our secondary experiments. LT tends to focus on token-specific characteristics at the early steps and then turns to find sequence-specific ones at the late steps, especially in deep settings. Meanwhile, stable and smooth behaviors in the early iteration are significant. Although there are some practical limitations, as we mentioned in this paper, we believe our lazy transition is a novel perspective to reconsider the models based on iterative refinements in sequence modeling.

[10] As mentioned before, LT can inherit the recurrent inductive bias of UT for handling varying complexity.

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Rewon Child David Luan Dario Amodei Ilya Sutskever Alec Radford, Jeffrey Wu. 2020. [GPT-2] Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, 1(May):1–7.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep Equilibrium Models. In *Advances in Neural Information Processing Systems*.

Ankur Bapna, Mia Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033, Brussels, Belgium. Association for Computational Linguistics.

Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ale s Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2015. 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.

Zewei Chu, Wang Hai, Kevin Gimpel, and David McAllester. 2017. Broad context language modeling as reading comprehension. In *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference*, volume 2, pages 52–57.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. 2017. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning*.

Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6292–6299.

Mostafa Dehghani, Stephan Gouws, Jakob Uszkoreit Łukasz, Kaiser Google, and Brain Google Brain. 2019. UNIVERSAL TRANSFORMERS. In *7th International Conference on Learning Representations, ICLR 2019 - Conference Track Proceedings*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Stanisław Jastrz Ebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. 2018. Residual connections encourage iterative inference. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. DEPTH-ADAPTIVE TRANSFORMER. In *8th International Conference on Learning Representations, ICLR 2020 - Conference Track Proceedings*.

Carlos Escolano, Marta R. Costa-jussà, José A. R. Fonollosa, and Mikel Artetxe. 2021. Multilingual machine translation: Closing the gap between shared and language-specific encoder-decoders. In

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 944–948, Online. Association for Computational Linguistics.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*.

Alex Graves. 2016. Adaptive Computation Time for Recurrent Neural Networks.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. 2018. Hyperbolic attention networks. In *7th International Conference on Learning Representations, ICLR 2019 - Conference Track Proceedings*.

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie Yan Liu, and Wei Ying Ma. 2016. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pages 820–828.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2017. Tracking the world state with recurrent entity networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, volume 2015-Janua, pages 190–198.

Łukasz Kaiser and Ilya Sutskever. 2016. Neural GPUs learn algorithms. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Diederik P Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, pages 6156–6175.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *8th International Conference on Learning Representations, ICLR 2020 - Conference Track Proceedings*.

Zhouhan Lin, Minwei Feng, Cicero Nogueira Dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Mauro Mezzini. 2018. Empirical study on label smoothing in neural networks. In *WSCG 2018 - Short papers proceedings*.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 news translation task submission. In *Proceedings of the Fourth Conference on Machine Translation*, pages 314–319, Florence, Italy. Association for Computational Linguistics.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context *. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. 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.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Théophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. 2018. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*.

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*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Samuel L Smith, David H.P. Turban, Steven Hamblin, and Nils Y Hammerla. 2017. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Ashish Vaswani, Google Brain, 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*, pages 5998–6008.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. 2016. Towards AI-complete question answering: A set of prerequisite toy tasks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1112–1122.

Shijie Wu, Alexis Conneau, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Emerging cross-lingual structure in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. 2019. Tied Transformers: Neural Machine Translation with Shared Encoder and Decoder. volume 33, pages 5466–5473.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho Jui Hsieh. 2020. Large batch optimization for deep learning: Training bert in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020 - Conference Track Proceedings*.

Wojciech Zaremba and Ilya Sutskever. 2014. Learning to Execute. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Biao Zhang, Ankur Bapna, and Orhan Sennrich, Rico;Firat. 2021. Share or not? Learning to schedule language-specific capacity for multilingual translation. In *9th International Conference on Learning Representations, ICLR 20201- Conference Track Proceedings*, August, pages 1–19.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 19–27.

A Introduction to Recurrence

Concretely, token-wise recurrent models recur over depth for each token. The computational bound of recurrence is not the number of tokens in the sequence but is the maximum number of refinements made to representations, i.e., the pre-defined maximum number of steps. By contrast, recurrent neural networks recur over positions for the sequence. The comparison is presented in Figure 1.

B Optimization Challenge in the Combination of UT and GRU

We observe an optimization challenge. In practice, we find this combination fails due to gradient

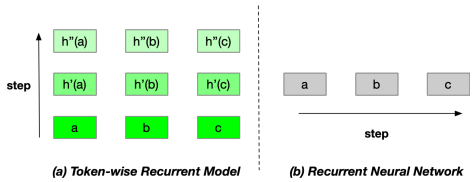


Figure 1: Recurrent neural network and token-wise recurrent model. h' and $h"$ are hidden representations. Given the sequence input 'a b c', recurrent neural network recurs over positions, whereas token-wise recurrent model recurs over each token at each step with shared parameters for each step and each token.

explosions. We suspect naive GRU is not compatible well with UT, and UT cannot converge due to GRU. Inspired by (Smith et al., 2017), we impose an orthogonal constraint to all the kernels in GRU. Orthogonal kernels ensure that the unique quality of representations is preserved because an orthogonal matrix preserves the dot product of vectors, as well as their L2 distances, and is an isometry of the Euclidean space, which helps to retain all the information from UT. Moreover, it makes optimization stable in our experiment. Therefore, we add a kernel constraint or update strategy (Cisse et al., 2017) to all kernels:

$$W \leftarrow (1+\beta)W - \beta(WW^T)W \quad (8)$$

where $\beta = 0.001$ is suggested by (Cisse et al., 2017). This kernel constraint ensures that all the kernels stay close to the manifold of orthogonal matrices after each update.

C More Comparison

Lazy Transition vs. GRU As aforementioned, the computational procedure of lazy transition is similar to GRU (Cho et al., 2015; Chung et al., 2014), but the backend is significantly different. Concretely, despite the difference of the recurrence mechanism, GRU emerges segment-level representations including all the computed representations before the current position, whereas our lazy transition only emerges token-level representations informed by the sequence representation of previous steps. Also, they have different motivations. GRU aims to learn a representation for a sequence by accumulating the information from all the tokens, whereas LT is encouraged to ponder the importance of every step for different tokens and then to reach the *relaxed* equilibrium. Meanwhile, compared to GRU, which forgets the previously computed state via the reset gate, our lazy transition dismisses the newly computed refinement.

Lazy Transition vs. Adaptive Computation Methods for adaptive computation like ACT (Graves, 2016) and Adaptive-depth Transformer (Elbayad et al., 2020) learn a generator to output a probability of exiting based on the step output. These methods are agonistic for the model's convergence because these methods do not consider the information flow from the input to the corresponding output. By contrast, our method leverages the model's convergence and applies CKA to ponder the correspondence and interdependence between inputs and outputs.

Lazy Transformer vs. UT, DEQ, and ALBERT Our work is parallel to UT (universal transformer) (Dehghani et al., 2019), DEQ (Bai et al., 2019), and ALBERT (Lan et al., 2020). We share the idea of recurrence over depth. However, we have three main differences: *1)* previous methods require step encodings in the model, whereas we consider step identification in our lazy transition without explicit step encodings; *2)* we consider the significance of a step for different tokens, whereas previous methods refine tokens equally at a step; *3)* we consider the local and token-level equilibrium in addition to the sequence-level equilibrium. On the other hand, from the view of halting refinement, compared to UT, LT does not need ACT (Adaptive Computation Time) (Graves, 2016). $\sigma_i^{t_i}$ dynamically prevents the update from the further refinement. By contrast, UT with the combination of ACT and step encodings dynamically computes the halting probability for the current step by employing a probability generator. From the perspective of UT, we believe the lazy transition is a good alternative to the combination of step encodings and ACT. Also, in our empirical study and experiments, LT can run a large number of steps to improve the performance significantly, i.e., training a deep model, whereas UT obtains limited improvements from a large number of steps.

D LAMBADA Language Modeling

We attempt the LAMBADA (Paperno et al., 2016) task, which is a dataset for language modeling tasks but within a broad context, to evaluate our model on language modeling tasks of varying complexity. The goal of the LAMBADA task is to predict the target word of the target sentence, based on a

narrative passage. Human subjects are easily able to guess the target word if they are exposed to the narrative passage but cannot accurately guess the target word if they only see the target sentence preceding the target word. The task is challenging because models cannot simply rely on local context but must understand the information in the broader context, which means that models have to genuinely understand the broad context in natural language.

In this test, we only use the setting of the standard set-up as language modeling that is more challenging. For the reading comprehension setting proposed by (Chu et al., 2017), we leave this extensive set-up for further experiments. Following the instructions (Parikh et al., 2016), we download the dataset from the official repository of LAMBADA$\diamond$, and then we train the model to predict the next word as a general language modeling task on the training dataset but only predict the target word at test time. Pre-training-based methods (Radford et al., 2018; Brown et al., 2020; Alec Radford, 2020) yield SOTA in this test.

E Case Study for Equilibrium Phenomenon

E.1 Visualization for Base NTM Model

We conduct a case study ($En \rightarrow De$) for the equilibrium phenomenon by inputting a random mid-length sentence from the test set: "The system is fitted with coloured LEDs, which are bright enough that drivers can easily see the lights, even when the sun is low in the sky." We use the CKA exam Eq.6 for this study, and we observe that:

- As presented in Figure 2 (a,b), in base-LT, the importance of a step differs from one token to the others in both the encoder and decoder because CKA scores (also $\sigma_i^{t_i}$ in Eq.7) are different. It confirms the effectiveness of our lazy transition.

- As presented in Figure 2 (a,b), base-LT has a smooth process to find the *relaxed* equilibrium in the encoder and decoder because each token representation does not change dramatically. On the contrary, as presented in Figure 2 (c,d), base-UT is hard because CKA scores dramatically change at the very early step for all tokens, e.g., from step 1 to step 2, the average of absolute difference is 0.221, which is only 0.105 in base-LT.

- We confirm base-UT tries to find a global direction for all tokens at every step because Figure 2 (c,d) shows a similar CKA score (importance) for each token. However, base-LT does not show such "global" effect in Figure 2 (a,b), which proves our *relaxed* equilibrium.

- Interestingly, we find that, as presented in Figure 2 (c,d), base-UT seems only to get significant benefits from the early steps because CKA scores at the late steps are too large. However, CKA does not consider scaling. Probably, these steps scale representations into an acceptable range.

E.2 Visualization for UT without Step Encodings

In Figure 3, base-UT w/o SE shows a similar behavior to base-UT, but it is more irregular in zero-shot inferring.

E.3 Visualization for UT + ACT

In Figure 4, base-UT + ACT shows a similar behavior to base-UT. Intuitively, ACT has no power to control the importance of a step throughout the iteration. Note that in our experiment base-UT + ACT has a general large gradient norm. It is potentially problematic in practice. Interestingly, this is the only configuration that the best performance precisely exists at the training step.

E.4 Visualization for UT + GRU

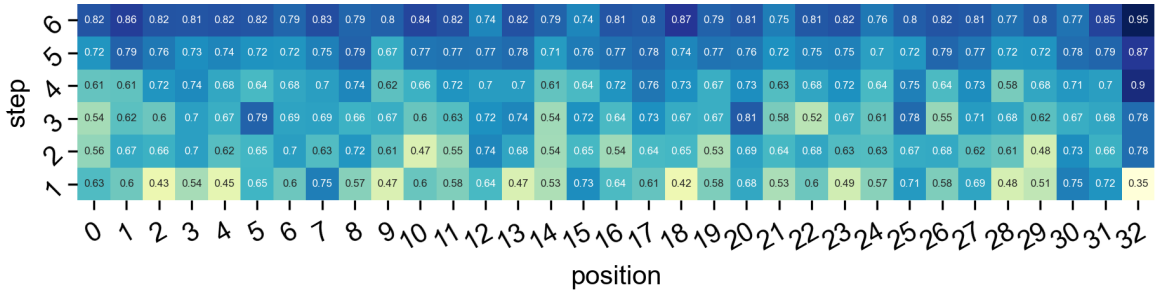
In Figure 5, similar to ACT, GRU has no power to control the importance of a step for tokens throughout the iteration. Meanwhile, we suspect base-UT + GRU fails in the test of zero-shot inferring due to the unstable refinement at the very early step. It is even more irregular than base-UT.

E.5 Visualization for Deep NTM Model

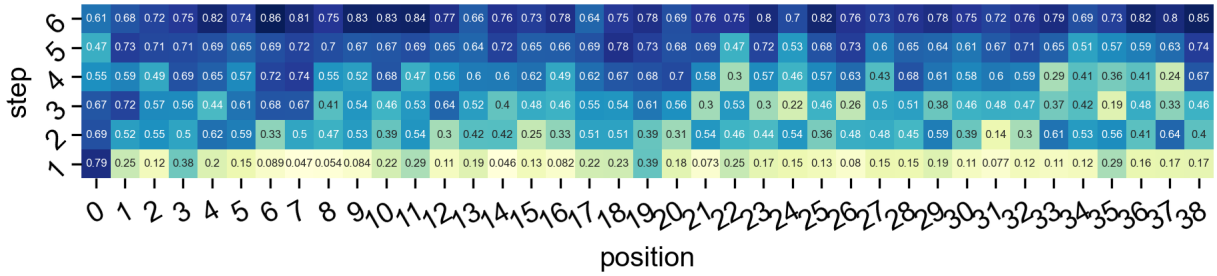
As presented in Figure 6 (a), in the encoder of 20-UT, CKA scores significantly change at the early step and are similar at every step throughout the iteration. By contrast, in Figure 6 (b) and Figure 7, tokens smoothly reach their local equilibriums, and then all the tokens search the *relaxed* equilibrium synchronously. Again, LT does not show the "global" effect throughout the iteration, which proves our *relaxed* equilibrium.

F Zero-shot Inferring

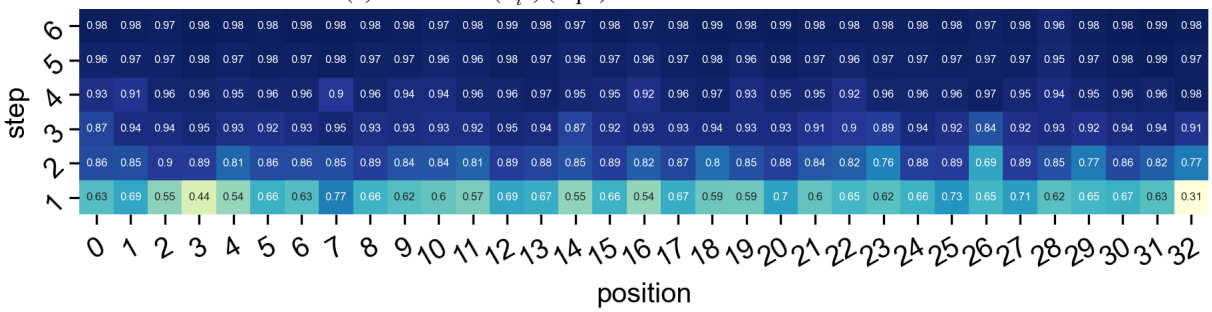
We list all the results of zero-shot inferring in Table 6 for base models and Table 7 for deep models.



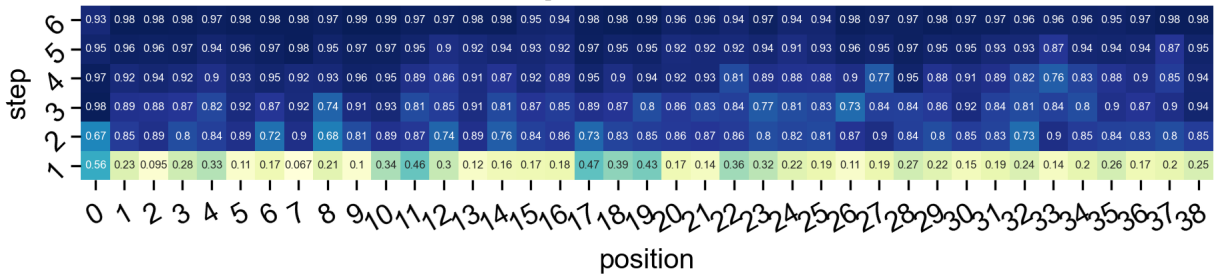
(a) CKA exam ($\sigma_i^{t_i}$) (Eq.6) for the encoder of base-LT.



(b) CKA exam ($\sigma_i^{t_i}$) (Eq.6) for the decoder of base-LT.



(c) CKA exam (Eq.6) for the encoder of base-UT.



(d) CKA exam (Eq.6) for the decoder of base-UT.

Figure 2: Exam of equilibrium phenomenon for base models.

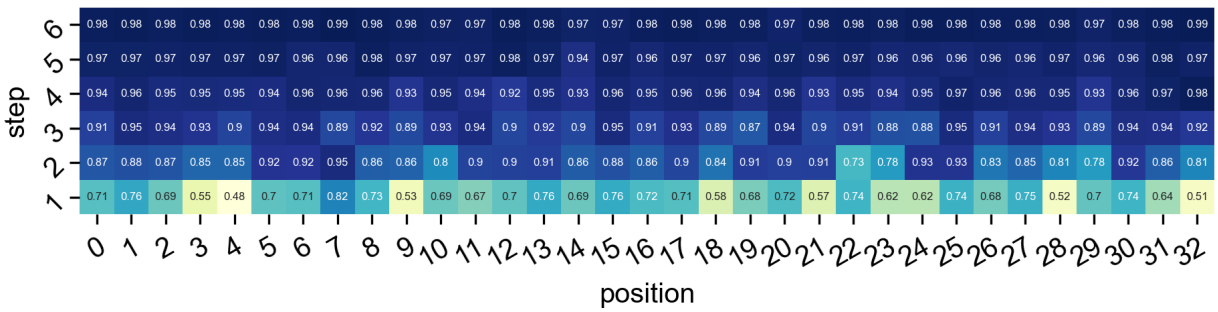


Figure 3: CKA exam (Eq.6) for the encoder of base-UT w/o SE.

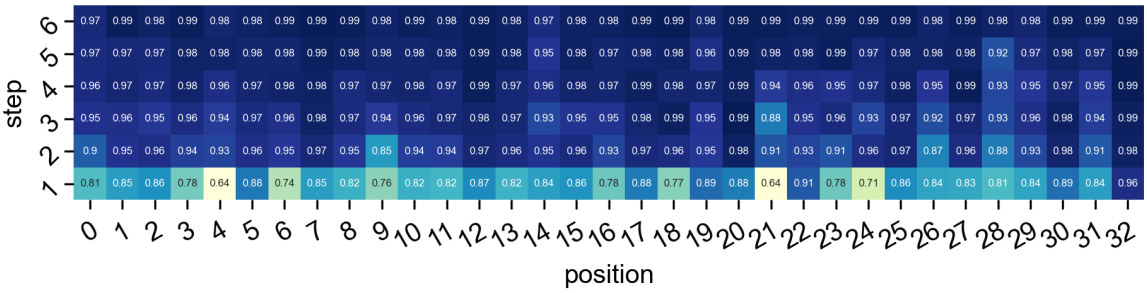


Figure 4: CKA exam (Eq.6) for the encoder of base-UT + ACT.

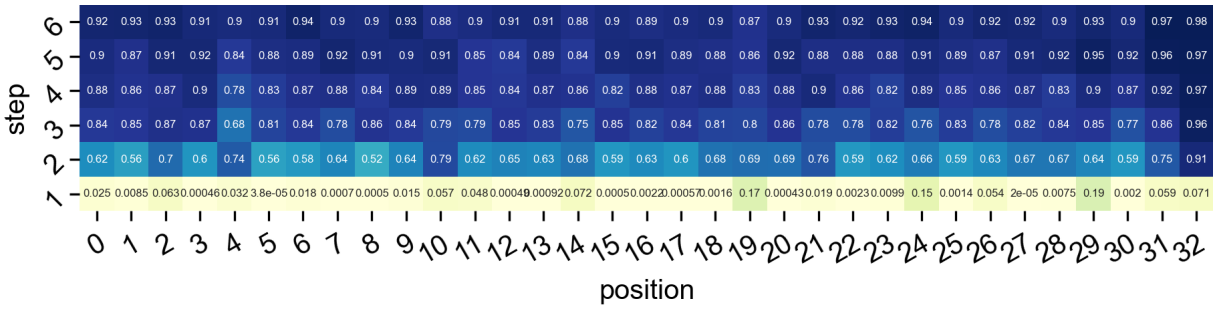
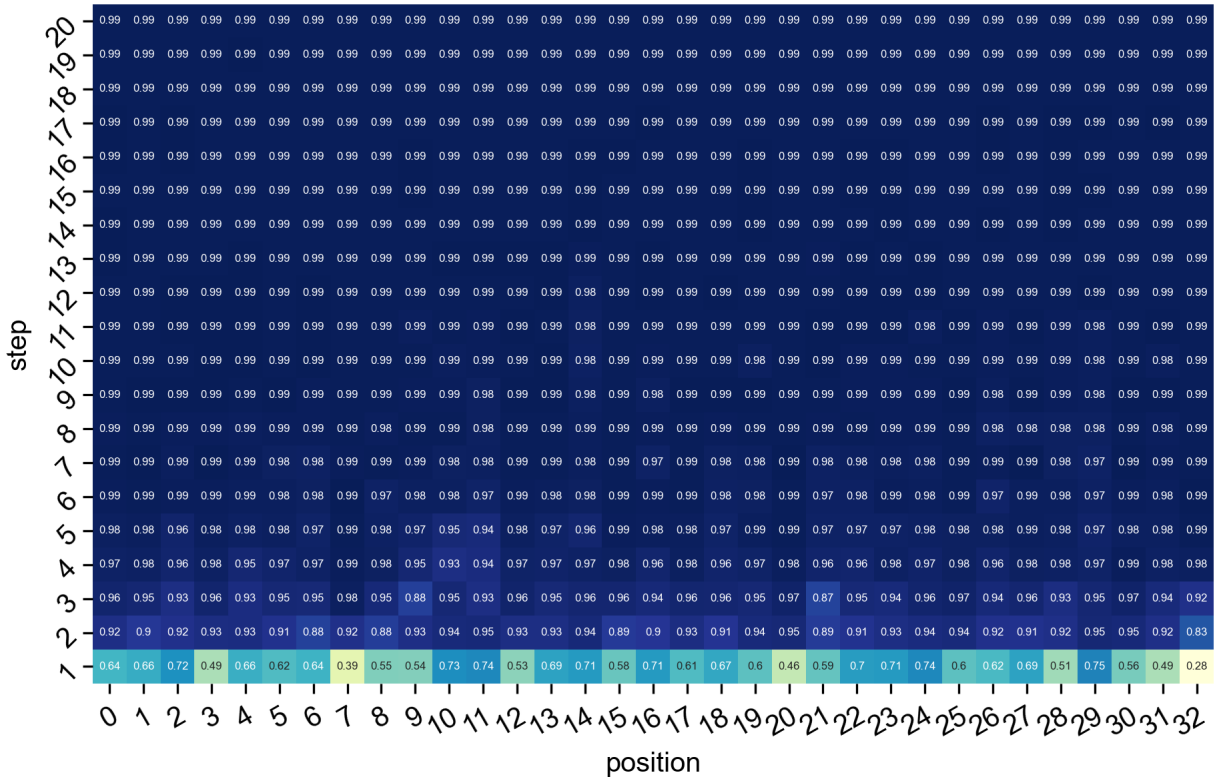
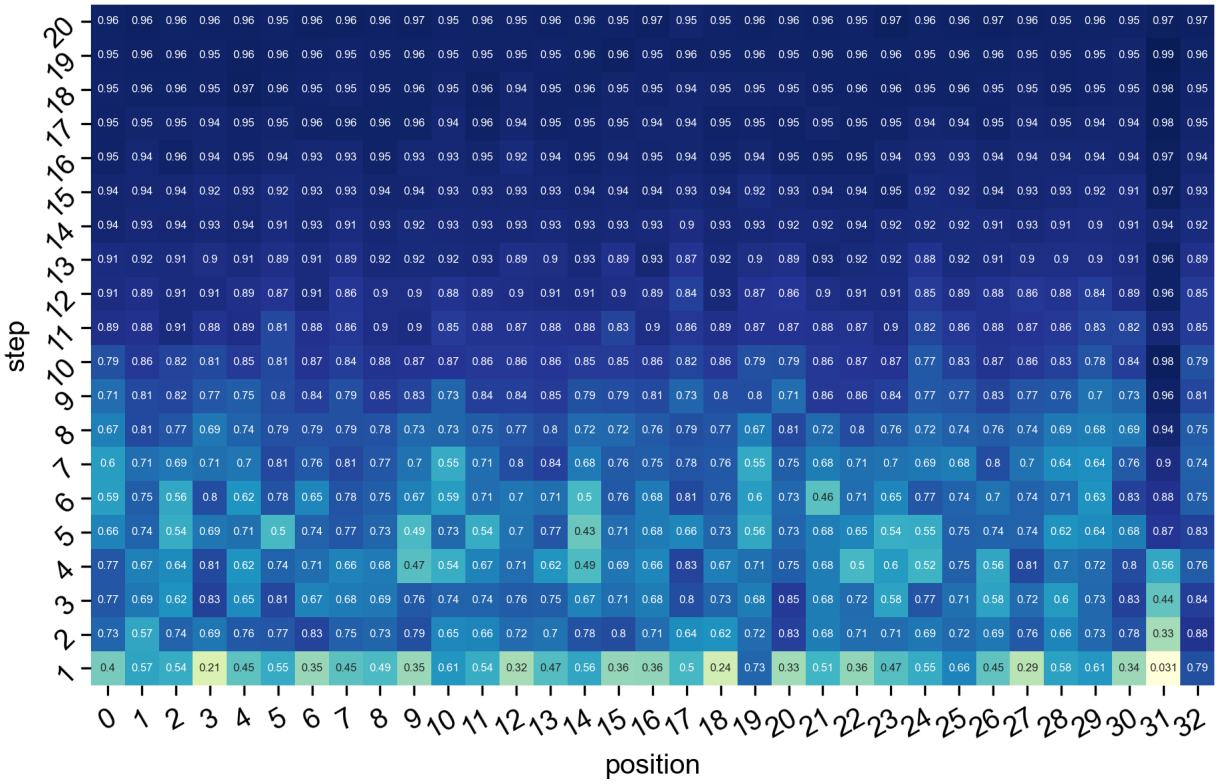


Figure 5: CKA exam (Eq.6) for the encoder of base-UT + GRU.



(a) CKA exam (Eq.6) for the encoder of 20-UT.



(b) CKA exam ($\sigma_i^{t_i}$) (Eq.6) for the encoder of 20-LT.

Figure 6: Exam of equilibrium phenomenon for deep models.

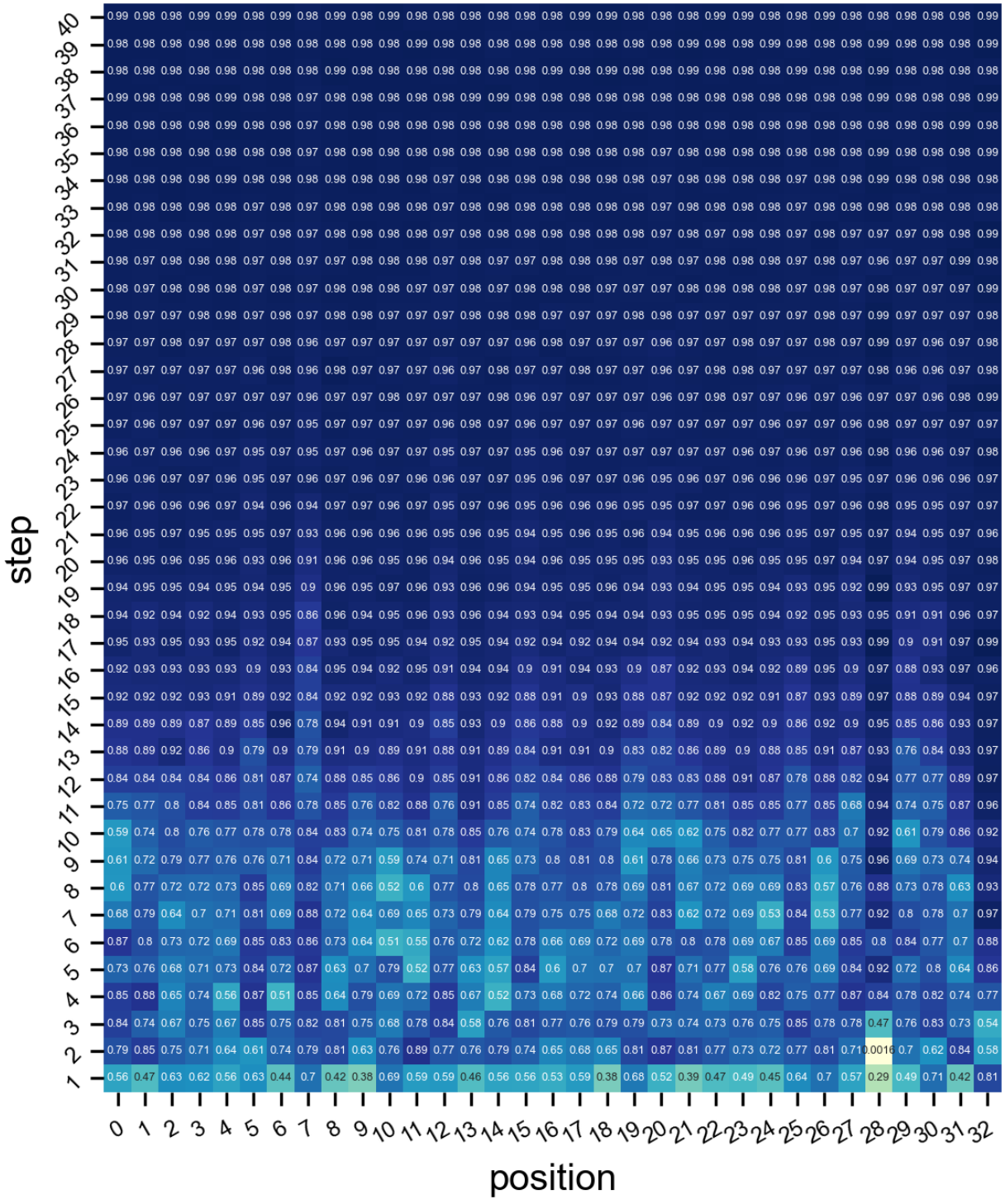


Figure 7: CKA exam ($\sigma_i^{t_i}$) (Eq.6) for the encoder of 40-LT.

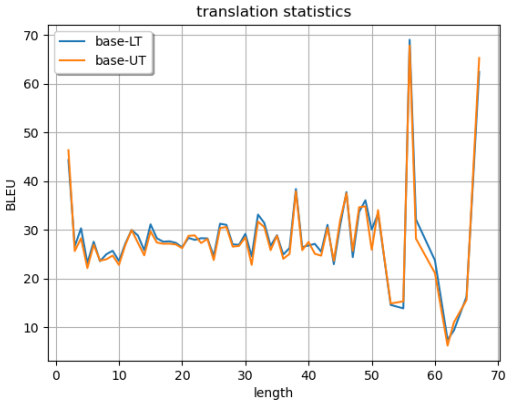


Figure 8: Translation statistics for base-LT and base-UT.

When connecting to the visualization earlier, we conjecture that using the CKA exam is a good solution to probe training problems and understand the learning behavior. From all the tables, we can conclude that:

- Compared to base models, deep models can generally find a strong equilibrium, which results in higher avg and lower std.
- Compared to UT, LT can oscillate around a stronger equilibrium and is more stable (higher avg and lower std).
- Although GRU seems to be able to balance two iteration steps, it cannot help UT find a strong equilibrium and is unstable.
- We suspect unstable training (a large gradient norm) is the main problem for base-UT + ACT in NMT. The best performance precisely exists at the training step, and the performance is constant after the training step. We attribute this phenomenon to the halting mechanism of ACT.

G Translation Statistics

We find both base-LT and base-UT are not stable when translating sentences longer than 50 words. As presented in Figure 8, the BLEU score varies from 5 to 60 for different sentences. We will conduct further experiments to probe this problem.

H Source

We list all the links of datasets, tools, and other sources in Table 8.

Encoders / Decoder	1	2	3	4	5	6	7	8	9	10	11	12	20
base-UT:$\{6,6\} \implies \{X,X\}$ (nonzero: $avg:22.16, std:6.32$)													
1	0.0	0.0	0.0	4.79	5.35	5.33	5.31	5.70	5.54	3.37	3.77	3.59	2.83
2	0.0	0.0	0.0	15.57	18.44	19.34	17.54	16.97	19.09	20.91	19.30	15.86	3.41
3	0.0	0.0	0.0	21.94	22.96	23.55	24.37	24.96	22.91	22.21	21.76	22.34	13.25
4	0.0	0.0	0.0	24.17	27.67	24.63	24.73	23.16	22.53	24.61	24.67	22.86	13.00
5	0.0	0.0	0.0	26.55	27.32	25.49	25.84	26.41	25.85	25.84	23.28	23.28	17.89
6	0.0	0.0	0.0	26.47	24.66	**28.73**	25.92	23.97	24.02	23.88	24.17	22.03	23.74
7	0.0	0.0	0.0	25.24	28.89	28.95	28.58	25.34	25.35	23.98	24.04	24.88	22.29
8	0.0	0.0	0.0	24.04	27.70	28.95	27.00	24.04	24.04	24.28	25.25	24.61	23.67
9	0.0	0.0	0.0	22.68	27.32	26.60	27.00	25.85	23.73	23.84	23.84	26.65	25.72
10	0.0	0.0	0.0	22.18	27.63	26.66	26.37	28.61	27.93	27.34	22.30	22.98	21.93
11	0.0	0.0	0.0	22.12	27.02	28.92	28.22	28.69	25.10	23.58	22.02	24.46	19.59
12	0.0	0.0	0.0	21.25	29.57†	29.22	26.96	27.7	23.58	25.03	23.62	23.92	18.30
20	0.0	0.0	0.0	14.95	21.40	21.60	22.48	22.33	22.01	23.51	21.27	20.05	14.21
base-UT w/o SE:$\{6,6\} \implies \{X,X\}$ (nonzero: $avg:21.04, std:7.94$)													
1	0.0	0.0	0.0	2.51	5.67	8.59	9.41	7.03	6.66	5.93	6.39	8.19	4.60
2	0.0	0.0	0.0	2.72	22.14	20.06	21.38	21.46	20.01	21.76	20.53	20.53	16.54
3	0.0	0.0	0.0	4.43	24.37	25.11	25.24	26.08	22.44	23.88	21.92	19.93	17.68
4	0.0	0.0	0.0	13.88	23.42	24.89	24.94	23.74	22.13	22.92	20.50	20.52	14.99
5	0.0	0.0	0.0	13.88	24.16	23.95	23.56	24.19	22.86	22.12	20.66	18.34	19.69
6	0.0	0.0	0.0	7.47	23.48	**27.85**	27.82	29.65	30.69†	25.53	25.33	22.41	21.61
7	0.0	0.0	0.0	3.33	25.68	28.08	29.53	31.05	31.87	27.91	26.60	25.72	18.71
8	0.0	0.0	0.0	3.92	25.15	28.51	27.94	30.25	30.85	23.4	25.42	25.44	16.81
9	0.0	0.0	0.0	1.63	23.34	28.69	27.94	28.19	26.98	26.65	26.81	25.27	17.34
10	0.0	0.0	0.0	1.56	23.36	28.53	27.96	25.43	24.85	26.21	27.03	26.87	17.30
11	0.0	0.0	0.0	0.09	23.36	28.12	27.94	28.72	26.06	24.55	25.00	26.87	18.24
12	0.0	0.0	0.0	0.09	21.15	28.35	24.63	26.94	27.53	24.66	25.13	24.94	18.41
20	0.0	0.0	0.0	0.17	23.88	25.57	25.04	21.95	23.91	22.69	22.69	24.69	16.45
base-UT + GRU:$\{6,6\} \implies \{X,X\}$ (nonzero: $avg:17.45, std:9.13$)													
1	0.0	0.0	0.0	0.0	3.62	3.6	3.4	5.25	1.91	2.35	1.18	1.10	0.63
2	0.0	0.0	0.0	0.0	18.29	14.23	10.26	13.78	9.85	8.25	6.72	5.51	0.32
3	0.0	0.0	0.0	0.0	21.76	23.13	19.13	20.64	19.89	19.66	14.88	7.22	0.29
4	0.0	0.0	0.0	0.0	24.22	23.82	24.61	26.17	26.23	22.54	19.28	12.34	0.48
5	0.0	0.0	0.0	0.0	23.32	22.97	26.22	22.41	26.62	19.83	18.33	10.09	0.68
6	0.0	0.0	0.0	0.0	22.93	**26.59**	23.17	23.58	26.07	20.44	18.54	16.33	0.38
7	0.0	0.0	0.0	0.0	22.93	22.74	23.23	22.95	28.11	20.63	17.46	13.82	0.38
8	0.0	0.0	0.0	0.0	23.49	22.79	23.28	22.95	28.04	23.83	21.23	8.25	0.56
9	0.0	0.0	0.0	0.0	23.62	22.79	23.79	25.57	28.04	28.57	21.29	10.37	0.78
10	0.0	0.0	0.0	0.0	22.82	22.51	22.30	26.00	29.04†	25.69	18.25	6.18	0.66
11	0.0	0.0	0.0	0.0	22.82	21.16	22.78	25.51	28.35	26.44	18.38	13.41	0.62
12	0.0	0.0	0.0	0.0	23.60	24.01	25.27	28.45	26.54	22.05	17.90	18.13	0.58
20	0.0	0.0	0.0	0.0	21.73	25.38	24.03	27.91	26.49	26.03	15.08	18.67	0.75
base-UT + ACT:$\{6,6\} \implies \{X,X\}$ (nonzero: $avg:23.81, std:6.75$)													
1	0.0	0.0	0.0	0.42	5.13	6.10	5.95	6.03	6.03	6.03	6.03	6.03	6.03
2	0.0	0.0	0.0	10.53	16.97	12.20	14.22	14.22	14.22	14.22	14.22	14.22	14.22
3	0.0	0.0	0.0	19.60	26.37	25.27	25.90	25.90	25.90	25.90	25.90	25.90	25.90
4	0.0	0.0	0.0	19.63	26.27	26.75	26.03	26.03	26.03	26.03	26.03	26.03	26.03
5	0.0	0.0	0.0	16.79	25.68	27.30	27.65	27.65	27.65	27.65	27.65	27.65	27.65
6	0.0	0.0	0.0	18.14	26.35	**29.11**†	29.11	29.11	29.11	29.11	29.11	29.11	29.11
7	0.0	0.0	0.0	17.26	25.79	28.33	27.96	27.96	27.96	27.96	27.96	27.96	27.96
8	0.0	0.0	0.0	20.50	25.59	27.65	27.50	27.50	27.50	27.50	27.50	27.50	27.50
9	0.0	0.0	0.0	20.39	25.16	27.65	27.98	27.98	27.98	27.98	27.98	27.98	27.98
10	0.0	0.0	0.0	17.69	24.90	27.96	27.94	27.94	27.94	27.94	27.94	27.94	27.94
11	0.0	0.0	0.0	21.32	25.04	27.70	27.99	27.99	27.99	27.99	27.99	27.99	27.99
12	0.0	0.0	0.0	20.03	24.04	27.18	27.38	27.38	27.38	27.38	27.38	27.38	27.38
20	0.0	0.0	0.0	22.10	22.44	26.31	25.94	25.94	25.94	25.94	25.94	25.94	25.94
base-LT:$\{6,6\} \implies \{X,X\}$ (nonzero: $avg:24.52, std:5.03$)													
1	0.0	0.0	0.0	1.30	10.21	10.22	10.53	10.60	10.38	11.64	11.62	10.39	9.84
2	0.0	0.0	0.0	15.05	20.18	19.81	18.88	20.09	21.31	21.20	20.35	20.64	22.50
3	0.0	0.0	0.0	19.05	23.52	24.10	24.32	24.21	24.05	23.91	23.92	25.43	26.66
4	0.0	0.0	0.0	24.28	26.23	26.88	24.13	23.61	24.77	25.04	24.74	25.17	27.77
5	0.0	0.0	0.0	21.35	25.41	25.93	26.16	24.96	25.38	25.53	25.53	25.94	25.63
6	0.0	0.0	0.0	24.46	26.35	**29.81**	27.45	24.41	26.95	24.49	25.16	24.27	24.30
7	0.0	0.0	0.0	24.83	29.45	26.61	27.27	27.08	27.21	24.75	24.75	24.81	27.51
8	0.0	0.0	0.0	24.74	28.70	27.86	27.03	27.48	28.32	27.72	27.30	24.81	24.51
9	0.0	0.0	0.0	26.67	29.02	25.39	28.88	28.04	27.99	27.99	29.03	25.70	28.29
10	0.0	0.0	0.0	26.07	29.64	28.76	29.63	28.04	28.09	25.44	25.44	25.39	25.16
11	0.0	0.0	0.0	26.13	31.47†	28.35	29.35	29.61	26.07	26.04	30.72	24.83	25.16
12	0.0	0.0	0.0	26.13	31.15	28.29	26.16	26.26	26.07	26.14	26.04	24.78	25.17
20	0.0	0.0	0.0	23.87	28.11	28.07	28.15	26.50	26.45	26.45	25.70	25.70	25.80

Table 6: Zero-shot inferring for base models. Training steps are in bold. † denotes the best performance.

Encoders / Decoder	1	2	3	4	5	6	7	8	9	10	11	12	20
20-UT:$\{20, 6\} \implies \{X, X\}$ (nonzero: $avg : 26.03, std : 4.62$)													
15	0.0	0.0	0.0	24.47	27.83	26.76	28.61	28.01	29.28	28.54	26.48	22.72	9.91
16	0.0	0.0	0.0	24.44	27.42	27.30	27.47	27.91	29.82	26.01	29.25	23.37	17.26
17	0.0	0.0	0.0	23.87	25.45	27.27	27.47	28.66	28.26	25.13	24.55	27.06	15.64
18	0.0	0.0	0.0	24.02	26.28	26.82	27.48	29.11	27.90	28.62	24.92	27.3	15.73
19	0.0	0.0	0.0	24.02	26.24	27.02	27.48	29.18	29.43	29.55	28.38	28.42	13.27
20	0.0	0.0	0.0	23.97	26.20	**29.69**	27.48	27.45	29.43	29.55	28.24	28.83	16.72
21	0.0	0.0	0.0	23.55	26.14	27.23	26.92	28.23	30.08	28.92	28.48	27.60	14.29
22	0.0	0.0	0.0	22.93	25.88	27.27	26.76	28.29	29.50	28.92	30.67	29.99	15.76
23	0.0	0.0	0.0	23.65	26.77	26.36	27.56	30.67	30.35	29.29	29.85	29.99	14.42
24	0.0	0.0	0.0	24.28	26.63	25.95	29.75	30.60	30.53	28.97	29.05	28.54	12.55
25	0.0	0.0	0.0	22.02	26.92	26.91	29.96	30.45	29.81	28.79	28.98	28.43	12.92
26	0.0	0.0	0.0	22.13	26.86	27.19	30.08	30.96†	30.48	28.69	29.05	29.00	11.54
27	0.0	0.0	0.0	21.42	24.70	27.15	27.37	25.55	25.55	24.39	24.18	24.41	10.05
20-LT:$\{20, 6\} \implies \{X, X\}$ (nonzero: $avg : 28.89, std : 1.65$)													
15	0.0	0.0	0.0	26.45	27.31	27.83	27.96	29.87	29.85	30.56	30.26	30.26	31.56
16	0.0	0.0	0.0	26.41	27.31	27.62	28.0	29.72	29.85	30.15	30.19	30.15	31.49
17	0.0	0.0	0.0	26.17	26.41	26.82	27.92	29.72	29.85	30.15	30.19	30.19	31.49
18	0.0	0.0	0.0	26.17	27.26	26.82	27.82	29.82	29.85	30.15	30.19	30.19	32.05†
19	0.0	0.0	0.0	26.17	27.26	26.85	27.82	29.82	29.55	30.15	30.19	30.19	30.80
20	0.0	0.0	0.0	26.17	26.98	**30.54**	27.65	29.82	29.55	29.6	30.03	30.75	30.80
21	0.0	0.0	0.0	25.78	26.90	27.63	27.90	29.45	29.55	29.98	30.49	30.75	30.29
22	0.0	0.0	0.0	26.23	26.90	27.63	27.90	29.45	29.45	29.98	30.49	30.63	30.29
23	0.0	0.0	0.0	25.61	26.90	27.59	26.96	29.56	29.98	29.98	30.49	30.63	31.26
24	0.0	0.0	0.0	25.61	26.90	27.59	28.65	30.10	29.98	29.45	29.89	30.49	31.26
25	0.0	0.0	0.0	25.61	26.86	27.73	28.54	29.05	29.87	29.45	29.89	30.49	31.26
26	0.0	0.0	0.0	25.33	26.86	27.69	27.42	28.76	29.35	29.45	29.89	30.49	30.89
40	0.0	0.0	0.0	25.29	26.86	27.25	27.84	28.52	28.7	28.85	28.77	28.99	30.89
40-LT:$\{40, 6\} \implies \{X, X\}$ (nonzero: $avg : 29.31, std : 1.54$)													
35	0.0	0.0	0.0	26.34	27.92	28.36	28.89	29.55	29.58	28.90	28.90	29.88	31.59
36	0.0	0.0	0.0	26.34	27.92	27.82	28.35	29.55	29.58	28.90	28.82	29.88	31.59
37	0.0	0.0	0.0	26.34	27.92	27.82	28.35	29.55	29.58	28.90	28.82	29.88	31.59
38	0.0	0.0	0.0	26.24	27.92	28.70	28.35	29.55	29.58	28.90	28.82	29.88	31.59
39	0.0	0.0	0.0	26.24	27.92	30.05	28.35	29.65	29.58	28.82	29.88	29.88	31.59
40	0.0	0.0	0.0	26.24	28.82	**31.05**	29.29	30.18	29.58	28.82	29.88	29.88	31.59
41	0.0	0.0	0.0	25.55	28.82	31.05	28.77	30.18	29.58	28.82	29.88	29.88	31.59
42	0.0	0.0	0.0	25.55	28.82	31.05	28.77	31.11	29.58	28.82	29.88	29.88	31.59
43	0.0	0.0	0.0	25.55	28.82	31.05	28.77	31.11	29.58	28.93	29.88	29.88	31.59
44	0.0	0.0	0.0	25.55	28.82	31.05	28.77	31.11	30.53	28.93	29.88	29.88	31.59†
45	0.0	0.0	0.0	25.55	28.29	31.05	28.27	31.11	30.43	28.93	29.88	29.88	31.48
46	0.0	0.0	0.0	25.55	28.29	31.05	28.27	31.11	30.43	28.93	29.88	29.88	31.48
50	0.0	0.0	0.0	25.46	28.29	31.05	28.27	30.64	30.54	29.88	29.88	29.88	31.48

Table 7: Zero-shot inferring for deep models. Training steps are in bold. † denotes the best performance.

Item	Links
WMT 2014	http://www.statmt.org/wmt14/translation-task.html
WMT 2016	http://www.statmt.org/wmt16/translation-task.html
FLoRes	https://github.com/facebookresearch/flores
sacreBleu	https://github.com/mjpost/sacrebleu
Moses tokenizer	https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl
fastBPE	https://github.com/glample/fastBPE
SemEval'17	https://alt.qcri.org/semeval2017/task2/
XNLI	https://github.com/facebookresearch/XNLI
BooksCorpus	https://yknzhu.wixsite.com/mbweb
WikiExtractor	http://opus.nlpl.eu
FAIR	https://github.com/pytorch/fairseq/tree/main/examples/translation
tensor2tensor	https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/research/universal_transformer.py
CKA tool	https://colab.research.google.com/github/google-research/google-research/blob/master/representation_similarity/Demo.ipynb
LAMB	https://github.com/ymcui/LAMB_Optimizer_TF

Table 8: Links of source.