HAL
open science

How Many Layers and Why? An Analysis of the Model Depth in Transformers

Antoine Simoulin, Benoît Crabbé

To cite this version:

Antoine Simoulin, Benoît Crabbé. How Many Layers and Why? An Analysis of the Model Depth in Transformers. Association of Computational Linguistics (student), 2021, Bangkok, Thailand. hal-03601412

HAL Id: hal-03601412

https://hal.science/hal-03601412v1

Submitted on 8 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

How Many Layers and Why? An Analysis of the Model Depth in Transformers

Antoine Simoulin[1,2] **Benoît Crabbé**[2]
[1]Quantmetry [2]University of Paris, LLF
asimoulin@quantmetry.com
bcrabbe@linguist.univ-paris-diderot.fr

Abstract

In this study, we investigate the role of the multiple layers in deep transformer models. We design a variant of ALBERT that dynamically adapts the number of layers for each token of the input. The key specificity of ALBERT is that weights are tied across layers. Therefore, the stack of encoder layers iteratively repeats the application of the same transformation function on the input. We interpret the repetition of this application as an iterative process where the token contextualized representations are progressively refined. We analyze this process at the token level during pre-training, fine-tuning, and inference. We show that tokens do not require the same amount of iterations and that difficult or crucial tokens for the task are subject to more iterations.

1 Introduction

Transformers are admittedly over-parametrized (Chen et al., 2020; Hou et al., 2020; Voita et al., 2019). Yet the role of this over-parametrization is not well understood. In particular, transformers consist of a fixed number of stacked layers, which are suspected to be highly redundant (Liu et al., 2020) and to cause over-fitting (Fan et al., 2020; Zhou et al., 2020). In this paper we provide a study on the role of the multiple layers traditionally used.

The mechanism of transformer layers is often compared to intuitive NLP pipelines (Tenney et al., 2019). Starting with the lower layers encoding surface information, middle layers encoding syntax and higher layers encoding semantics (Jawahar et al., 2019; Peters et al., 2018). Transformers progressively refine the features, which become more fine-grained at each iteration (Xin et al., 2020). However, ALBERT (Lan et al., 2020) highlights that it is possible to tie weights across layers and repeat the application of the same function. Consequently, we hypothesize that it is the number of layer applications that gradually abstracts the surface information into semantic knowledge.

To better study the transformation of token representations across layers, we propose a variant of ALBERT. Our model implements the key specificity of weights tying across layers but also dynamically adapts the number of layers applied to each token. Since all layers share the same weight, we refer to the application of the layer to the hidden states as an *iteration*.

After reviewing the related work (Section 2), we detail the model and the training methodology in Section 3. In particular, we encourage our model to be parsimonious and limit the total number of iterations performed on each token. In Section 4, we analyze iterations of the model during pre-training, fine-tuning and inference.

2 Related Work

Adapting the transformer depth is an active subject of research. In particular, deep transformer models are suspected to struggle to adapt to different levels of difficulty. While large models correctly predict difficult examples, they over-calculate simpler inputs (Liu et al., 2020). This issue can be addressed using *early-stopping*: some samples might be sufficiently simple to classify using intermediate features. Some models couple a classifier to each layer (Zhou et al., 2020; Liu et al., 2020; Xin et al., 2020). After each layer, given the classifier output, the model either immediately returns the output or passes the sample to the next layer. Exiting too late may even have negative impacts due to the network "over-thinking" the input (Kaya et al., 2019).

Ongoing research also refines the application of layers at the token level. Wang and Kuo (2020) build sentence embeddings by combining token representations from distinct layers. Elbayad et al. (2020) and Dehghani et al. (2019) successfully use

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop, pages 221–228
August 5–6, 2021. ©2021 Association for Computational Linguistics

dynamic layers depth at the token level for full transformers (encoder-decoder). However, to the best of our knowledge, our attempt is the first to apply such mechanism to encoder only transformers and to provide an analysis of the process.

3 Method

In this Section, we detail the model architecture, illustrated in Figure 1, and pre-training procedure.

3.1 Model architecture

We use a multi-layer transformer encoder (Devlin et al., 2019) which transforms a context vector of tokens $(u_1 \cdots u_T)$ through a stack of L transformer encoder layers (Eq. 1, 2). We use weight tying across layers and apply the same transformation function at each iteration (Lan et al., 2020).

$$h_t^0 = W_e u_t + W_p \tag{1}$$

$$h_t^n = \text{layer}(h_t^{n-1}) \quad \forall n \in [1, L] \tag{2}$$

For the first layer, W_e is the token embedding matrix, and W_p the position embedding matrix.

We augment the model with a halting mechanism, which allows dynamically adjusting the number of layers for each token (Eq. 3 to 8). We directly adapted this mechanism from Graves (2016). The main distinction with the original version is the use of a transformer model instead of a recurrent state transition model. The mechanism works as follow: at each iteration n, we add the following operations after Eq. 2. We assign a probability to stop p_t^n for each token at index t (Eq. 3). Given this probability, we compute an update weight λ_t^n (Eq. 4), which we use to compute the final state as the linear convex combination between the previous and current hidden state (Eq. 5).

$$p_t^n = \sigma\left(W_h h_t^n + b_n\right) \tag{3}$$

$$\lambda_t^n = p_t^n \text{ if } n < N_t, R_t \text{ elif } n = N_t, \text{ else } 0 \tag{4}$$

$$h_t^n = \lambda_t^n h_t^n + (1 - \lambda_t^n) h_t^{n-1} \tag{5}$$

With σ the sigmoid function. We define the remainder R_t and the number of iterations for the token at index t, N_t with:

$$R_t = 1 - \sum_{l=1}^{N_t - 1} p_t^l. \quad N_t = \min_{n'} \sum_{n=1}^{n'} p_t^n \geq 1 - \epsilon \tag{6}$$

As soon as the sum of the probability becomes greater than 1, the update weights λ_t^n are set to 0 and the token is not updated anymore (Eq. 4). A small ϵ factor ensures that the network can stop after the first iteration (Eq. 6).

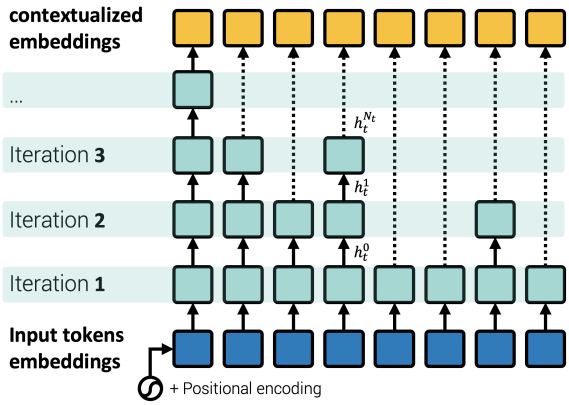

Figure 1: As in ALBERT model, tokens are transformed through the iterative application of a transformer encoder layer. Our model key specificity is the application of the halting mechanism, which dynamically adjusts the number of iterations for each token.

3.2 Pre-training objective

During the pre-training phase, we train the model with the sentence order prediction (*sop*) — the task introduced in Lan et al. (2020) that classifies whether segments from the input sequence follow the original order or were swapped — and the masked language model task (*mlm*) (Devlin et al., 2019). We also encourage the network to minimize the number of iterations by directly adding the ponder cost into ALBERT pre-training objective. Given a length T input sequence $\mathbf{u}$, Graves (2016) defines the *ponder cost* $\mathcal{P}(\mathbf{u})$ as:

$$\mathcal{P}(\mathbf{u}) = \sum_{t=1}^{T} N_t + R_t \tag{7}$$

We define the final pre-training loss as the following sum:

$$\hat{\mathcal{L}} = \mathcal{L}_{sop} + \mathcal{L}_{mlm} + \tau \mathcal{P} \tag{8}$$

where τ is a *time penalty* parameter that weights the relative cost of computation versus error.

3.3 Datum and infrastructure

We follow the protocol from ALBERT and pre-train the model with BOOKCORPUS (Zhu et al., 2015)

and English Wikipedia. We reduce the maximum input length to 128 and the number of training steps to 112,500[1]. We use a lowercase vocabulary of size 30,000 tokenized using SentencePiece. We train all our models on a single TPU v2-8 from Google Colab Pro[2] and accumulate gradients to preserve a 4,096 batch size. We optimize the parameters using LAMB with a learning rate at 1.76e-3.

4 Experiments

We now analyze our iterative model properties during pre-training (Section 4.1) and fine-tuning (Section 4.2). We start by describing the setup for each of the subtasks.

***mlm* task** We generate masked inputs following ALBERT n-gram masking. We mask 20% of all WordPiece tokens but do not always replace masked words with the `[MASK]` token to avoid discrepancy between pre-training and fine-tuning. We effectively replace 80% of the masked position with `[MASK]` (`[MASK/MASK]`), 10% with a random token (`[MASK/random]`), and keep the original token for the last 10% (`[MASK/original]`).

***sop* task** We format our inputs as "`[CLS]` x_1 `[SEP]` x_2 `[SEP]`". In 50% of the case the two segments x_1 and x_2 are effectively consecutive in the text. In the other 50%, the segments are swapped.

Ponder cost We fix the time penalty factor τ empirically such that the ponder penalty represents around 10% of the total loss. To estimate the ponder cost, we discard the remainder, as $R \ll N$ for sufficient values of N. Given Eq. 7, the ponder cost then corresponds to the total number of iterations in the sentence, which is given by $l \times T$, with T the number of tokens in the sequence and l the average iterations per token. We observe that ALBERT base loss converges to around 3.5. We calibrate τ such that $\tau\mathcal{P} \approx 0.35 \approx \tau \times l \times T$. We train distinct models, listed in Table 1, that we calibrate such that their average number of iterations per token l is respectively 3, 6, and 12. We refer to these models as respectively *tiny*, *small* and *base*.

[1]As emphasized in https://github.com/google-research/bert, longer sequences are computationally expensive. To lighten the pre-training process, they advise using 128 sentence length and increase the length to 512 only for the last 10% of the training to train the positional embeddings. In this work, we only perform the first 90% steps as we are not looking for brute force performances.

[2]https://colab.research.google.com/

4.1 Analysis of the pre-training

Analysis of the iterations We pre-train models with various configurations and observe the model mechanisms during the pre-training in Table 1.

Models	*tiny*	*small*	*base*
τ	1e-3	5e-4	2.5e-4
Max iterations	6	12	24
mlm (Acc.)	55.4	57.1	57.4
sop (Acc.)	80.9	83.9	84.3
All tokens	3.8	7.1	10.0
All unmasked tokens	3.5	6.5	9.2
`[MASK/MASK]`	5.8	10.9	16.0
`[MASK/random]`	5.8	10.9	16.0
`[MASK/original]`	4.0	7.4	10.5
`[CLS]`	6.0	12.0	22.5
`[SEP]`	2.5	7.6	8.4

Table 1: Average number of iterations given token types during the pre-training. For each model, we report a mean number of iterations on our development set, at the end of the pre-training.

We observe that the `[CLS]` token receives far more iterations than other tokens. This observation is in line with Clark et al. (2019) who analyze BERT attention and report systematic and broad attention to special tokens. We interpret that the `[CLS]` token is used as input for the *sop* task and aggregates a representation for the entire input. On the contrary, `[SEP]` token benefits from usually few iterations. Again, this backs the observation emerging from the analysis of attention that interprets `[SEP]` as a no-op operation for attention heads (Clark et al., 2019).

We also observe an interesting behavior from the `[MASK]` which also benefits from more iterations than average tokens. As for the `[CLS]` token, we interpret that these tokens are crucial for the *mlm* task. Looking further, we observe that `[MASK/random]` and `[MASK/MASK]` number of iterations is greater than `[MASK/original]`. In this case, although all tokens are targeted in the *mlm* task, `[MASK/random]` and `[MASK/MASK]` are obviously more difficult to identify[3].

The model seems to have an intuitive mechanism

[3]During inference, the model cannot make the distinction between `[MASK/original]` and unmasked tokens. However, we observe in Table 1 that the two token types have a distinct mean number of iterations. We believe this is due to the distribution of the `[MASK]` tokens. Indeed, we follow the procedure from ALBERT and use *n-gram* masking. Therefore, `[MASK/original]` tokens tend to appear in the context of `[MASK]` tokens. This specific context increases the mean number of iterations.

and distributes iterations for tokens that are either crucial for the pre-training task or present a certain level of difficulty. This also appears in line with *early-exit* mechanisms cited in Section 2, that adapt the number of layers, for the whole example, to better scale to each sample level of difficulty.

Natural Fixed point We now analyze *how* the token's hidden states evolve during our model iterative transformations. At each iteration n, the self-attentive mechanism (Vaswani et al., 2017) computes the updated state $n+1$ as a weighted sum of the current states. This introduces a cyclic dependency as every token depends on each other during the iterative process. As convergence within a loopy structure is not guaranteed, we encourage the model to converge towards a fixed point (Bai et al., 2019).

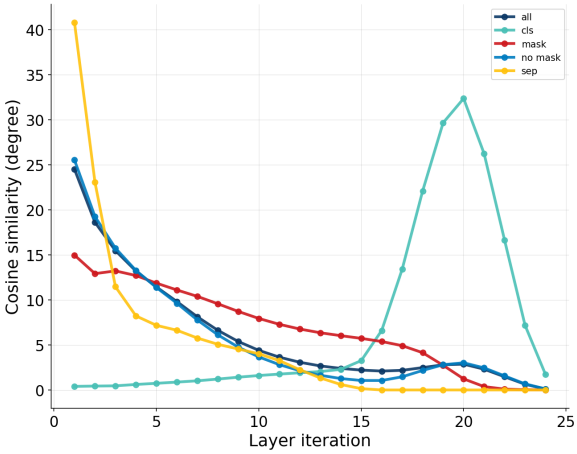

Figure 2: Evolution of the cosine similarity between hidden states h_t^n and h_t^{n+1} from two consecutive iterations. We use our *base* model and measure iterations on our development set, at the end of the pre-training.

We obtain this property "for free" thanks to our architecture specificity. Indeed at each iteration, the hidden state is computed as a convex combination of the previous n and current $n+1$ hidden state. The combination is controlled by λ_t^n (Eq. 5). If λ_t^n is closed to 0, then $h_t^n \approx h_t^{n+1}$ and by definition (Eq. 4, 6) λ_t^n will eventually be set to 0 at a certain iteration.

Figure 2 represents the evolution of the mean cosine similarity between two hidden states from two consecutive iterations h_t^n and h_t^{n+1}. The network indeed reaches a fixed point for every token. The `[SEP]` and tokens that are not masked converge quicker than `[MASK]` tokens. Finally, the `[CLS]` token oscillates during intermediate layers before reaching an equilibrium[4].

4.2 Application to downstream tasks

During the pre-training phase, the model focuses on tokens either crucial for the pre-training task or presents a certain level of difficulty. Now we study our model behavior during the fine-tuning on downstream syntactic or semantic tasks.

Control test To verify that our setup has reasonable performance, we evaluate it on the GLUE benchmark (Wang et al., 2019). Results from Table 2 are scored by the evaluation server[5]. As in Devlin et al. (2019), we discard results for the WNLI task[6]. For each task, we fine-tune the model on the train set and select the hyperparameters on the dev set using a grid search. We tune the learning rate between 5e-5, 3e-5, and 2e-5; batch size between 16 and 32 and epochs between 2, 3, or 4. To better compare our setup, we pre-train BERT and ALBERT model using our configuration, infrastructure and datum.

	Avg. Glue score
BERT-base	76.9
ALBERT-base	75.6
ALBERT-base + Adapt. Depth	75.2
ALBERT-small + Adapt. Depth	74.2
ALBERT-tiny + Adapt. Depth	72.6

Table 2: GLUE Test results, scored by the evaluation server but without the WNLI task. To facilitate the comparison, we reproduce BERT and ALBERT, with our pre-training dataset, infrastructure and configuration detailed in Section 3.2.

We present results on the test set in Table 2. As expected, the average score decreases with the number of iterations. Indeed, we limit the number of computation operations performed by our model. Moreover, we build our model on top of ALBERT, which share parameters across layers, thus reducing the number of parameters compared with the original BERT architecture. However, despite these additional constraints, results stay in a reasonable range. In particular, ALBERT-base with adaptative depth is very close to the version with a fixed depth.

[4]We present the Figures for other model configurations in Appendix A

[5]https://gluebenchmark.com/leaderboard

[6]See (12) from https://gluebenchmark.com/faq.

Probing tasks Conneau and Kiela (2018) introduce probing tasks, which assess whether a model encodes elementary linguistic properties. We consider semantic and syntactic tasks that do not introduce random replacements. In particular, a task that predicts the sequence of top constituents immediately below the sentence node (TopConst), a task that predicts the tense of the main-clause verb (Tense), and two tasks that predict the subject (resp. direct object) number in the main clause (SubjNum, resp. ObjNum).

	Tense	Subj Num	Obj Num	Top Const
punct (121k)	5.0	4.8	5.2	6.7
prep (101k)	4.6	4.6	5.4	6.2
pobj (98k)	4.5	4.6	5.4	5.8
det (86k)	4.5	4.6	5.1	6.1
nn (81k)	5.1	5.4	**5.8**	6.7
nsubj (80k)	**5.3**	**6.1**	**5.9**	**7.5**
amod (66k)	4.6	4.9	5.5	6.1
dobj (49k)	4.8	5.0	**5.9**	6.1
root (44k)	**5.9**	**6.1**	**6.2**	**7.9**
advmod (37k)	4.8	4.8	5.3	6.8
avg.	5.4	5.4	5.8	7.2
test Acc.	87.5	93.9	96.1	91.2
baseline Acc.	87.3	94.0	96.0	91.9

Table 3: Distribution of the iterations across token dependency types. We fine-tune our *base* model on each probing task. We then perform inference on the Penn Tree Bank dataset and report the number of iterations given token dependency types. The number in parentheses denotes the number of dependency tags. We only display the top 10 most frequent tags. We indicate in **bold** tags for which the number of iterations is above avg + std. We include a baseline accuracy which we obtain with the ALBERT-base version without an adaptative depth mechanism and therefore 12 iterations performed for each token.

In our setup, we fine-tune the model on the task train set and select the hyperparameters on the dev set using a grid search. We use a 5e-5 learning rate and fine tune the epochs between 1 to 5; we use a 32 batch size. Finally, we compare in Table 3 the number of iterations performed for each token on the Penn Tree Bank (Marcus et al., 1993) converted to Stanford dependencies[7,8].

We provide an accuracy baseline, obtained with the same setup but using ALBERT without the dynamic halting mechanism. As in the previous experiment, we observe that for these tasks, out model achieve competitive performances despite using less computational operations.

Although all tasks achieve significant and comparable accuracies, they all require a distinct global mean of iterations. The Tense task, which can be solved from the verb only, is completed in only 5.4 iterations, while the TopConst task, which requires to infer some sentence structure, is performed in 7.2 iterations. This suggests the model can adapt itself to the complexity of the task and globally spare unnecessary iterations.

Looking at the token level, as during the pre-training (Section 4.1), the iterations are unevenly distributed across tokens. The model seems to iterate more on tokens that are crucial for the task. For SubjNum, the subj tokens achieve the maximum number of iterations, while for the ObjNum task, the obj and root token iterates more. Similarly, all tasks present a high number of iteration on the main verb (root) that is crucial for each prediction.

5 Conclusion

We investigated the role of the layers in deep transformers. We designed an original model that progressively transforms each token through a dynamic number of iterations. We analyzed the distribution of these iterations during pre-training and confirmed the results obtained by analyzing the distribution of attention across BERT layers, particularly the specific behavior played by special tokens. Moreover, we observed that key tokens for the prediction task benefit from more iterations. We confirmed this observation during fine-tuning, where the tokens with a large number of iterations are also suspected to be key for achieving the task.

Our experiments provide a new interpretation path for the role of layers in deep transformer models. Rather than extracting some specific features at each stage, layers could be interpreted as the iteration from an iterative and convergence process. We hope that this can help to better understand the convergence mechanisms for transformers models, reduce the computational footprint or provide new regularization methods.

[7]Since we use sentence piece vocabulary, we assign to each piece the dependency tag from the whole token.

[8]We present the Tables for other model configurations in Appendix B

References

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 688–699.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020. Adabert: Task-adaptive BERT compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2463–2469. ijcai.org.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of bert's attention. *CoRR*, abs/1906.04341.

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3301–3310. PMLR.

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, Addis Ababa, Ethiopia, April 26-30, 2020*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6035–6044. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1499–1509. Association for Computational Linguistics.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4593–4601. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the*

57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 5797–5808. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Bin Wang and C.-C. Jay Kuo. 2020. SBERT-WK: A sentence embedding method by dissecting bert-based word models. *IEEE ACM Trans. Audio Speech Lang. Process.*, 28:2146–2157.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yukun Zhu, Ryan Kiros, Richard S. 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 *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27.

A Natural fixed point

We present here the evolution of the mean cosine similarity between two hidden states from two consecutive iterations for our *small* (Figure 3) and *tiny* (Figure 4) models. As presented in Section 3.2, we fix the maximum number of iterations at respectively 6 and 12 for the *tiny* and *small* models.

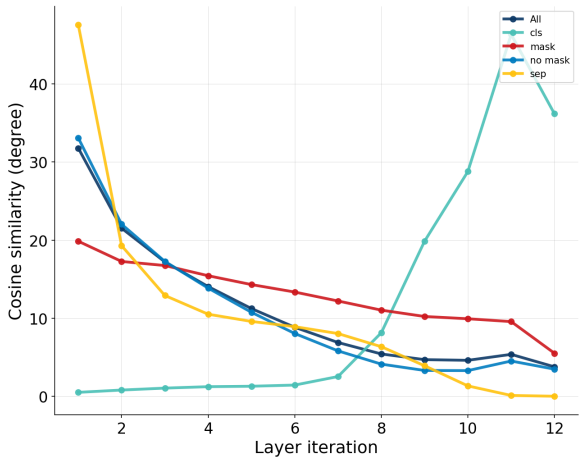

Figure 3: Evolution of the cosine similarity between hidden states h_t^n and h_t^{n+1} from two consecutive iterations. We use our *small* model and measure iterations on our development set, at the end of the pre-training.

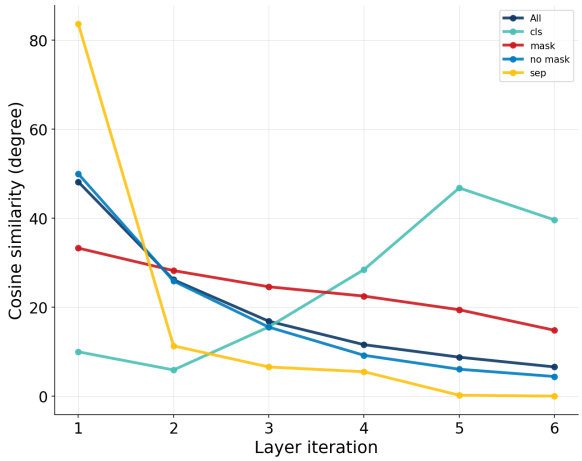

Figure 4: Evolution of the cosine similarity between hidden states h_t^n and h_t^{n+1} from two consecutive iterations. We use our *tiny* model and measure iterations on our development set, at the end of the pre-training.

B Probing tasks

We give here the probing tasks results from Section 4.2 with our *small* (Table 4) and *tiny* (Table 5) models.

	Tense	Subj Num	Obj Num	Top Const
punct (121k)	3.1	3.1	3.1	3.9
prep (101k)	2.9	2.9	3.0	3.6
pobj (98k)	2.9	3.0	3.1	3.5
det (86k)	2.7	2.8	2.7	3.6
nn (81k)	3.2	**3.5**	3.2	3.9
nsubj (80k)	**3.3**	**3.7**	**3.3**	**4.4**
amod (66k)	2.9	3.0	3.0	3.6
dobj (49k)	3.0	3.2	**3.4**	3.5
root (44k)	**3.6**	**3.6**	**3.5**	**4.6**
advmod (37k)	2.9	3.0	3.0	4.0
avg.	3.2	3.3	3.3	3.9
test Acc.	86.4	93.2	95.5	91.1
baseline Acc.	87.3	94.0	96.0	91.9

Table 4: Distribution of the iterations across token dependency types. We fine-tune our *small* model on each probing task. We then perform inference on the Penn Tree Bank dataset and report the number of iterations given token dependency types. The number in parentheses denotes the number of dependency tags. We only display the top 10 most frequent tags. We indicate in **bold** tags for which the number of iterations is above avg + std. We include a baseline accuracy which we obtain with the ALBERT-base version without an adaptative depth mechanism and therefore 12 iterations performed for each token.

	Tense	Subj Num	Obj Num	Top Const
punct (121k)	2.1	1.9	2.0	2.5
prep (101k)	2.0	1.7	2.0	2.3
pobj (98k)	2.0	1.8	2.0	2.2
det (86k)	1.9	1.7	1.8	2.3
nn (81k)	**2.2**	**2.0**	**2.0**	2.5
nsubj (80k)	**2.3**	**2.2**	**2.1**	**2.8**
amod (66k)	2.1	1.8	2.0	2.3
dobj (49k)	2.1	1.9	**2.1**	2.3
root (44k)	**2.4**	**2.1**	**2.3**	**2.9**
advmod (37k)	2.1	1.8	2.0	2.6
avg.	2.2	2.0	2.1	2.5
test Acc.	88.6	91.1	93.8	91.1
baseline Acc.	87.3	94.0	96.0	91.9

Table 5: Distribution of the iterations across token dependency types. We fine-tune our *tiny* model on each probing task. We then perform inference on the Penn Tree Bank dataset and report the number of iterations given token dependency types. The number in parentheses denotes the number of dependency tags. We only display the top 10 most frequent tags. We indicate in **bold** tags for which the number of iterations is above avg + std. We include a baseline accuracy which we obtain with the ALBERT-base version without an adaptative depth mechanism and therefore 12 iterations performed for each token.