General Paper Peer-Reviewed

A Peek Into the Memory of T5: Investigating the Factual Knowledge Memory in a Closed-Book QA Setting and Finding Responsible Parts

Tareq Alkhaldi[†], Chenhui Chu[†] and Sadao Kurohashi[†]

Recent research shows that Transformer-based language models (LMs) store considerable factual knowledge from the unstructured text datasets on which they are pre-trained. The existence and amount of such knowledge have been investigated by probing pre-trained Transformers to answer questions without accessing any external context or knowledge (also called closed-book question answering (QA)). However, this factual knowledge is spread over the parameters inexplicably. The parts of the model most responsible for finding an answer only from a question are unclear. This study aims to understand which parts are responsible for the Transformer-based T5 reaching an answer in a closed-book QA setting. Furthermore, we introduce a head importance scoring method and compare it with other methods on three datasets. We investigate important parts by looking inside the attention heads in a novel manner. We also investigate why some heads are more critical than others and suggest a good identification approach. We demonstrate that some model parts are more important than others in retaining knowledge through a series of pruning experiments. We also investigate the roles of encoder and decoder in a closed-book setting.

Key Words: *Analysis, Attention Components, Question Answering, Transformers*

1 Introduction

Neural language models (LMs) store knowledge from the unstructured text on which they are pre-trained (Roberts et al. 2020; Petroni et al. 2019; Jiang et al. 2020). Such a phenomenon of storing knowledge is interesting and valuable because LMs are trained on the unstructured text and are easy to query. Previous work investigated the existence and amount of such knowledge in Transformer-based models, such as BERT (Devlin et al. 2018) and T5 (Raffel et al. 2019), and how to effectively query models to find the stored knowledge.

Querying these models is a form of question answering (QA), and it is considered a direct interface to the knowledge stored in LMs. Given that using context to answer questions makes it difficult to know if the model has retrieved this knowledge from within or from the context, we

[†] Graduate School of Informatics, Kyoto University

focus on analyzing this phenomenon using closed-book (QA) (Section 2). This setting is used by Roberts et al. (2020) to feed T5 models open-domain questions with no context to measure the amount of knowledge stored. In open-book QA, models are either fed context that contains the answer (Rajpurkar et al. 2016) or allowed to use external knowledge sources (Chen et al. 2017). However, in closed-book QA, the model only depends on the question and "look up information" stored in its parameters to find the answer. Therefore, closed-book QA is essential to estimate the amount of stored knowledge. For example, only feeding "What is the capital of Japan?" is sufficient for the model to generate the answer.

Although the amount of knowledge stored after pre-training is estimated using closed-book QA (Roberts et al. 2020), how this knowledge is stored and which parts of the T5 model are responsible for reaching answers remains poorly understood. This study aims to understand how knowledge is stored inside T5 in a closed-book QA setting while also studying the importance and role of different parts of the model (Section 3). We also present head importance scoring methods for pruning the model to verify the distribution of knowledge (Section 4). Understanding the inner workings and the crucial parts of the model allows for several benefits, such as guided pruning, where the size of LMs is reduced while preserving learned knowledge, aiding better design choices, and providing hints about how to integrate external knowledge with LMs better.

The contributions of this study are three-fold:

- After a novel investigation of the components of attention, we find that the query component is the most important, as it is almost the only component that changes when back-propagation is calculated.
- We find that the encoder finds the embedding space of the desired answer entity, and the decoder generates the textual representation of that entity.
- We compare different attention head importance scoring methods on three datasets and find that the stored knowledge is not evenly distributed in the model.

2 Preliminary

2.1 Task

This study aims to understand how QA Transformer-based T5 models generate answers to factual questions in a closed-book setting. It also attempts to find whether parts of the model are more important than others in reaching answers and then identify them.

In a closed-book QA setting, the model is given a question and is expected to provide an answer without accessing context or external knowledge. The model must "memorize" the answers given

during the pre-training or fine-tuning. How Transformers work in this setting is principally different than other tasks. In other settings, the model generally uses attention to focus on some parts of the provided context. For instance, the model can extract answers from a context paragraph in a normal QA setting (Devlin et al. 2018) or align words between source and target languages as in translation tasks. However, because only the question is provided in a closed-book QA setting, the model can only focus on the question tokens and how it decides the answer is unclear.

2.2 Datasets

Following Roberts et al. (2020), we use three open-domain QA datasets: Natural Questions (NQ) (Kwiatkowski et al. 2019), WebQuestions (WQ) (Berant et al. 2013), and TriviaQA (TQA) (Joshi et al. 2017). NQ and WQ are both datasets of questions from web queries, while TQA is a collection of questions from quiz league websites. All questions are accompanied with context of different forms to help find the answer. However, to use the datasets as closed-book QA datasets, all context is ignored, and only the questions are considered.

Our analysis uses the T5 (Raffel et al. 2019) models provided by Roberts et al. (2020). In particular, we use three variants of T5: T5-Large, T5-11b, and T5v1.1-xxl. Details about them are explained in Section 4.3.3.

2.3 Transformer Architecture

We briefly describe the Transformer architecture (Vaswani et al. 2017), which comprises the T5 model while focusing on the attention mechanism therein.

The Transformer is a model architecture with blocks of two stacked sub-layers: multi-head self-attention and feed-forward (FF) networks. Given an input sequence $\mathbf{x} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^{d_{model}}$, for each attention head h, a query Q^h, a key K^h, and a value V^h are computed from the input $\mathbf{x}$ as follows:

$$\begin{aligned} Q^h &= \mathbf{x}W_q^h \\ K^h &= \mathbf{x}W_k^h \\ V^h &= \mathbf{x}W_v^h \end{aligned} \tag{1}$$

where W_q^h, W_k^h, and W_v^h are parameter matrices for the head h. Q^h, K^h and V^h matrices contain $\mathbf{q}^h$, $\mathbf{k}^h$, and $\mathbf{v}^h$ vectors of different input tokens. Then, the attention scores α^h are computed as:

$$\alpha^h = \text{softmax}(\frac{Q^h {K^h}^T}{\sqrt{d_k}}) \tag{2}$$

where d_k is the key dimensionality. Then, the contextualized Z^h is computed by scaling V^h by attention scores as follows:

$$Z^h = \alpha^h V^h \tag{3}$$

with Z^h containing $\mathbf{z}^h$ vectors of the input tokens. The heads are then combined to produce Z as follows:

$$Z = \text{concat}_{h=1}^{H}(Z^h)W_o \tag{4}$$

where $\text{concat}_{h=1}^{H}$ is a function that concatenates Z^h of all H heads and W_o is a parameter matrix.

Next, the group of attended vectors Z is passed to the next component, a FF network. Vaswani et al. (2017) proposed using a two-layer network with ReLU activation. The final model has N layers of such blocks. When using the block in the decoder instead of the encoder, an additional sub-layer of cross-attention is added after the self-attention. This additional sub-layer receives its K and V from the output of the encoder.

3 Understanding Transformer-Based T5 in Closed-Book QA

The main component of Transformers is the attention mechanism; therefore, we start by looking into its components. Then, we verify the findings, and finally investigate the role of the encoder/decoder as a whole.

3.1 Dissecting Attention Components

Initially, we look at the gradients (or error back-propagation) resulting from the mismatch between the model prediction and the golden answer. They represent where the model attempts to fix itself to predict the correct answer. Areas with high gradients represent important parts responsible for reaching the answer. We gather these gradients and attempt to find interesting patterns.

Gathering Gradients

We evaluate the model on the validation set of NQ while calculating loss and applying a backward pass to obtain the gradients *without updating the parameters*. The loss is calculated against the gold labels for all tokens in one go using the built-in loss function in the T5 implementation of the Transformers library (Wolf et al. 2020). We reduce the calculated gradients for

each example (consisting of a question and a golden answer) by averaging each parameter matrix. The attention parameter matrices reduced are:

$$W_q^h, W_k^h, W_v^h \in \mathbb{R}^{d_k \times d_{model}}$$
$$W_o \in \mathbb{R}^{H \cdot d_k \times d_{model}}$$

where W_q^h, W_k^h, W_v^h and W_o are from the attention sub-layer as explained in Section 2.3. For completeness, we also analyze the FF matrices:

$$W_{ff1}^T, W_{ff2} \in \mathbb{R}^{ff_{size} \times d_{model}}$$

where W_{ff1} and W_{ff2} are the learnable matrices of the two-layer FF networks of each block. We then average the reduced results from all the examples.

Visualizing Gradients

We use the T5-Large provided by Roberts et al. (2020) with the NQ dataset to gather gradients, as explained previously. Fig. 1 visualizes the reduced gradients of the three attention types: Encoder.SelfAttention, Decoder.SelfAttention and Decoder.CrossAttention (shortened to EncSA, DecSA, and DecXA, respectively), each having 24 layers. Each row represents a layer and consists of the 16 heads of T5-Large. We also visualize the FF parameters of the encoder and decoder.

Interestingly, as shown in Fig. 1, almost only W_q is changed in all attention types. While

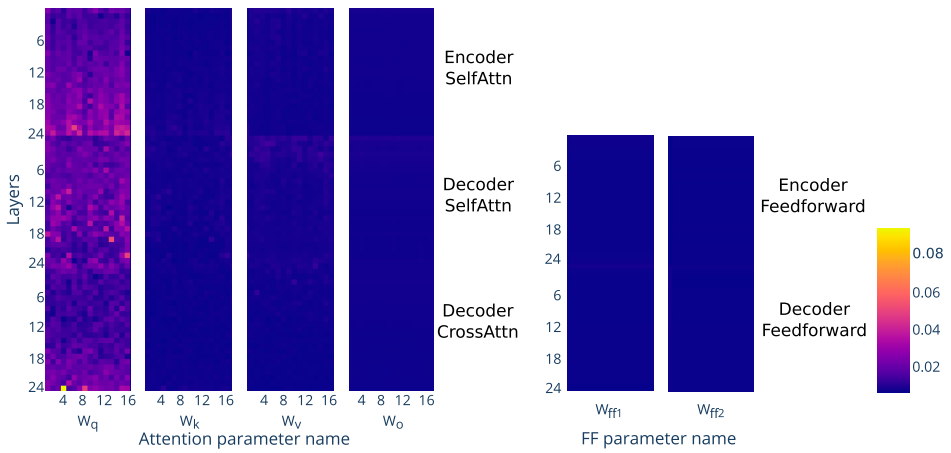

Fig. 1 Visualization of the averaged gradients of the attention and FF parameters for T5-Large over the validation split of the NQ dataset.

later layers are changed more in the EncSA and DecXA, the middle layers have the most change for DecSA. We notice minor changes in W_v from Fig. 1 and almost no change in the FF network parameters. This implies that, when the model predicts a wrong answer and the gradients are applied to fix the prediction, almost only **q** vectors are changed. We apply a similar analysis to other variants of T5 (T5-11b and T5v1.1-xxl) and find similar results.

Confirming the Importance of Q

Having shown the importance of Q for correcting wrong predictions, we conduct another experiment to confirm the importance and the applicability to correctly predicted answers. We add a normal noise $N(0.0, \sigma)$ to **q**, **k**, **v**, or FF vectors and observe the performance change, where σ is the standard deviation. A larger σ represents a more substantial noise.

We evaluate T5-large on the NQ dataset with noise added to one attention component or FF at a time. We show the results in Fig. 2 and compare them to the baseline of T5-large with no noise, which is 29.03 exact match (EM). Applying noise to **q** vectors degrades the performance substantially faster than other components, affirming the importance of the query component for wrongly and correctly predicted answers. The FF networks were most resilient to noise because of the drop-out training.

We assume the following: (1) The behavior of not changing W_k, W_v, or W_o is related to the fact that changes required to fix predictions are typically small. (2) A drastic change in the

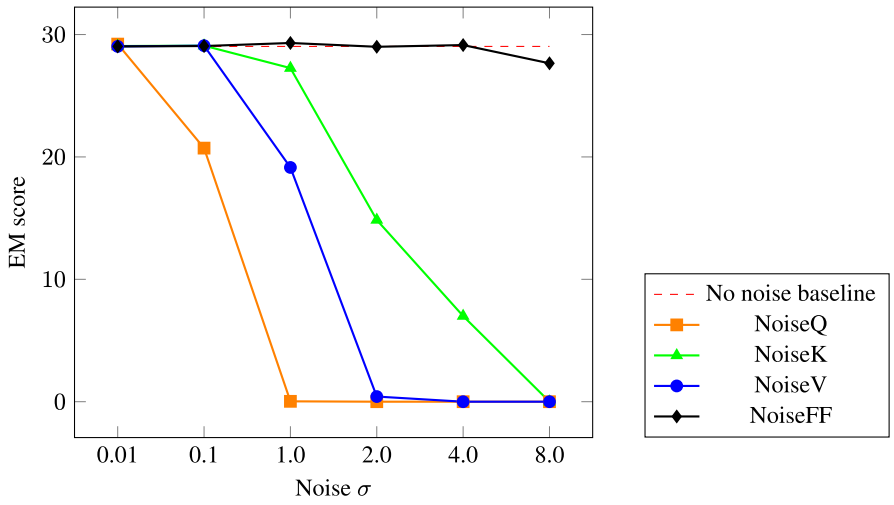

Fig. 2 Evaluation of T5-large on NQ dataset with noise of magnitude λ added to different attention components and FF output.

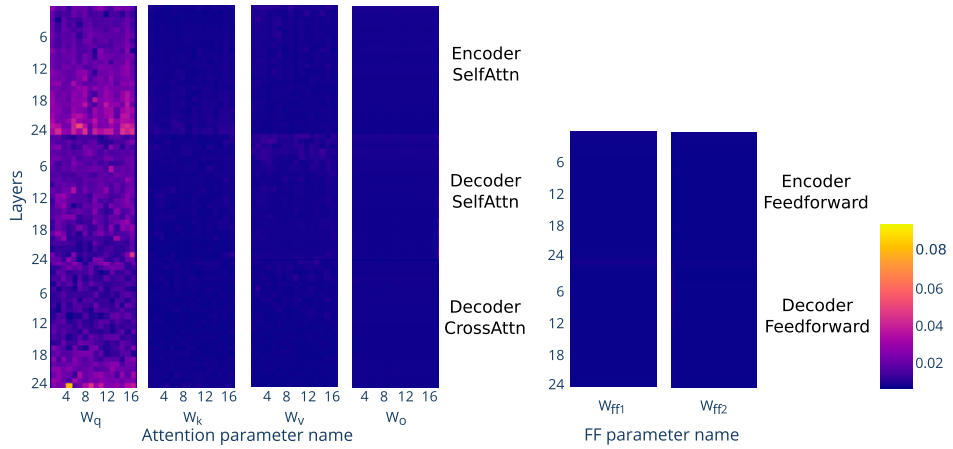

Fig. 3 Visualization of the averaged gradients of the attention parameters for T5-Large over the validation split of the NQ dataset after shuffling all target answers to random answers of a different named entity type.

expected answer invokes more changes in, for example, W_v. These assumptions can be answered by considering the type of predictions and golden answers.

We use SpaCy (Honnibal et al. 2020) to apply named entity recognition (NER) on all answers and compare the types of pairs of {prediction, gold} answers. The majority are of the same type (e.g., (GPE, GPE) or (DATE, DATE)). We then shuffle the golden answers, replacing each with a random golden answer of a different type. This would introduce many type mismatches, and the model would require bigger gradients/changes to fix the predictions. Then, we evaluate and gather the gradients again, as in Section 3.1 and show them in Fig. 3. Interestingly, the only change between Figs. 1 and 3 was slightly brighter W_q, with almost no change to the other attention parameters.

To verify whether W_k and W_v are changed during fine-tuning, we fine-tune a raw version of T5 on the NQ dataset and apply the gradient analysis explained earlier. While still only W_q was highly changed, at 50 and 250 steps, the decoder had significantly more gradients than the encoder, whereas, at 1k and 2k, the encoder had higher gradients. The results in Fig. 1 are after fine-tuning ends. The lack of change in W_k and W_v during fine-tuning implies that they were changed (and probably learned optimally) during the pre-training phase.

Difference Between Q and K

Although Q^h and K^h are computed by the same input and formula (except the weights), only

W_q receives high gradients. This gradient difference comes from the matrix multiplication order. To calculate the weight a^h, we do $Q^h {K^h}^T$, where the resulting matrix has rows that correspond to the queries and the columns to keys. Then, we take softmax on the last dimension, that is, by row. The softmax lets the row sum to 1.0, which would be the weights sum of our current query word attending to all other key words. The model learns this distinction between queries and keys and learns to change the queries rather than the keys.

How Knowledge Is Stored Inside T5

As experiments (Figs. 1 and 2) show that **q** vectors are the most crucial component for fixing predictions or reaching answers, we conclude that the following is how T5 stores knowledge.

In Eqs. 2 and 3, the role of **q** vectors is only to calculate a score that represents how much **v** to take from each token. Multiplying a scaling score with a vector does not change its direction, but only the norm of that vector. The **z** vectors are only composed from combining differently weighted **v** vectors (Eq. 3). For easier imagination, **v** vectors can be considered pointers or arrows of different lengths that point somewhere in the embedding space. When **z** vectors are combined cumulatively (over layers), they point to a place in the embedding space that represents the desired answer. In summary, Q^h is a map that guides the V^h arrows/pointers to this final point in the embedding space.

3.2 Role of Encoder/Decoder

Thus far, we have discussed attention head components as parts of the encoder–decoder model. Encoders and decoders usually have straightforward roles in other tasks, such as focusing on source/target languages and performing alignment in translation tasks, or selecting text spans in open-book QA. However, in a closed-book QA setting, their role is unclear. To investigate their role in a closed-book QA setting, we remove attention heads from one attention type at a time (e.g. EncSA) or FFs from the encoder or decoder and check the differences in predictions before and after a significant drop in performance. The procedure for removing heads is introduced in Section 4. We apply the same procedure for removing FFs. Comparing the results by removing EncSA heads and DecSA heads presented in Table 1, the role of the encoder is finding the correct entity while the decoder generates the textual representation of that entity. We did not witness special patterns when removing heads only from DecXA.

Table 2 shows a quantitative comparison of how predictions change before and after substantial pruning (with about 10 EM points drop). All predictions after pruning can either match predictions before pruning (EM), differ but with textual overlap, or become different with no

Prune Encoder.SelfAttention	Prune Decoder.SelfAttention
Southern Egypt	Jason Flemyng
Panama	Jason Fle
Sheryl Crow	Century City, Los Angeles
Elton John	Century City, California
Dob Robertson	a victim's dilemma
Robert Kelly	a victim of a distress signal
Subway	John Kramer
Eleven Madison Park	John C. J. Miller

Table 1 Patterns of the differences in predictions when only pruning EncSA vs DecSA. Each row's first and second lines show the prediction before and after pruning.

	EM	Textual Overlap	No Textual Overlap
Only pruning heads in encoder	36.70%	10.39%	52.91%
Only pruning FFs in encoder	43.32%	11.16%	45.51%
Only pruning heads in decoder	54.49%	23.74%	21.77%
Only pruning FFs in decoder	46.15%	25.51%	28.34%

Table 2 Quantitative comparison between the predictions before and after substantial pruning (not accuracy against gold answers).

overlap. Pruning the decoder heads does not change many predictions (high EM with predictions before pruning). It produces more wrong but textually overlapping predictions because the entity space has been decided in the encoder. However, harming the encoder heads results in predicting about 30% more non-textually overlapping predictions (i.e., different entities). Pruning the FFs in the encoder and decoder has a more negligible effect than pruning heads, but still follows the same trend.

4 Head Importance for Pruning

In a closed-book QA setting, the model must generate the correct answer with only access to the question tokens. The "knowledge" about the correct answer is within the model's parameters. We hypothesize that such "knowledge" is not distributed equally in the model, but there are areas that are significantly more important than others in retaining that knowledge.

To verify that some heads of the model are more important than others in retaining knowledge, we perform a series of pruning tests on attention heads according to different importance scoring

methods. We explore pruning least-important heads first or most-important heads first. The former provides a practical advantage where pruning models and reducing their parameters count while preserving their knowledge allows for faster and lighter models. Because FF networks do not store factual knowledge, as discussed in Section 3.1, we exclude them from further pruning experiments.

We explain the pruning settings and methods, visualize different importance maps, and compare them via pruning experiments on different datasets.

4.1 Pruning Flow

An importance score is given for each attention head. Then, the least/most 10% important heads are masked iteratively by adding a mask variable ξ^h following Michel et al. (2019), changing Eq. 3 as follows:

$$Z^h = \xi^h \alpha^h V^h \tag{5}$$

where the values of ξ^h are either 0 or 1. After each masking round, the model is re-evaluated and a new head importance map is calculated for the next masking round until we reach a threshold.

4.2 Importance Scoring Methods

For each of the three attention types (EncSA, DecSA, and DecXA), a head importance map is generated using one of the following methods:

4.2.1 Random

Heads are given random importance scores. This represents a counter hypothesis that knowledge is distributed equally in all of the model.

4.2.2 Gradients of Mask Gates (*attn*)

The importance score for a head is the gradient of the mask variable ξ^h. This method, introduced by Michel et al. (2019), focuses on the value of the attention score α^h. When the attention score is high, the gradient of ξ^h is also high; therefore, this method is called *attn*.

4.2.3 Summed Gradients of Attention Parameters (*grad_avg*)

Looking at the analysis result in Section 3.1, we introduce a new method of calculating the importance score by summing the reduced gradients of the parameter matrices in attention heads.

More specifically, for each head h, we calculate scores $grad^h_{avg}$ as follows:

$$grad^h_{avg} = \text{avg}(g(W^h_q)) + \text{avg}(g(W^h_k)) + \text{avg}(g(W^h_v))$$

where $g()$ is the function that obtains the gradients of a parameter matrix, and avg is a function that obtains the average of a matrix.[1] Since almost only W_q has high gradients among attention components (as shown in Fig. 1), adding the gradients of W_k and W_v has almost no effect on the final importance map. Therefore, we can consider that the importance of an attention head is equivalent to the importance of only the query component of that head. However, in this method, we still add the gradients of all components for completeness.[2]

4.2.4 Norm of $\mathbf{z}^h$ Vectors (*norm*)

This method does not depend on the gradients but on the norm of $\mathbf{z}^h$ vectors. Kobayashi et al. (2020) used the norm for analyzing alignment in machine translation. We use it as a scoring method; for each head h, the importance score $norm^h$ is calculated as follows:

$$norm^h = \sum_{t=1}^{n} ||\mathbf{z}^h_t||$$

where $\mathbf{z}^h_t$ is the contextualized vector $\mathbf{z}^h$ of token t, and $||\cdot||$ denotes the Euclidean norm.

4.3 Pruning Settings

4.3.1 Importance Sorting

- Local: To mask 10% of the heads, the importance scores are sorted per attention type *independently*. Then, the 10% are divided equally between the EncSA, DecSA, and DecXA attention types.
- Global: The maps of the three attention types are concatenated, and all the importance scores are sorted before selecting 10% to mask. Here, more heads might be masked from DecSA than EncSA, for example.

4.3.2 Normalization

- Layer normalization: Michel et al. (2019) normalize importance scores by layer using the ℓ_2 norm. We use this as the default.
- No layer normalization: We skip the layer normalization and denote the setting as *nln*.

[1] We also experimented with max for scoring; however, we excluded it because its performance was poor.

[2] We calculated the KL-divergence between importance maps with the gradients of all components, and maps of only W_q gradients. As the KL-divergence was 0.0001, confirming similar distributions, we exclude the latter maps.

4.3.3 Used Pre-trained Models

We use three variants of T5 released by Roberts et al. (2020)[3]: T5-large, T5-11b, and T5v1.1-xxl. They all have 24 layers but differ in several aspects, as shown in Table 3. T5v1.1 also differs from other models as it has no parameter sharing between embedding and classifier layers, and it was pre-trained on C4 (Raffel et al. 2019) only without mixing in the downstream tasks. It also uses GatedGelu instead of Relu for activations. Both T5v1.1-xxl and T5-11b have around 11 billion parameters.

All models had additional pre-training with salient span masking (Guu et al. 2020) before being fine-tuned. In this objective, salient spans (named entities and dates) are mined beforehand and then used in the pre-training as masks. A fine-tuned model is released for each of the three datasets (NQ, WQ, and TQA), except T5-large, which only has a version fine-tuned on NQ.

4.4 Visualizing Head Importance Maps

Fig. 4 shows the head importance maps generated with T5-large from the validation set of the NQ dataset for each scoring method. In "nln" all methods assign more importance to later layers in the EncSA, but have different assessments about the DecSA. For DecXA, all methods agree on the importance of certain heads in the last layer. Although there are differences in general between methods, the importance patterns are very similar across datasets, as shown in Fig. 5. This is likely because even though models were fine-tuned separately on the datasets, they all share the same pre-training, which is the actual source of the stored knowledge. This similarity is verified by pruning experiments in Section 4.5.3.

4.5 Experiments

We experiment with different scoring methods and settings with T5-large on the NQ dataset

Model	d_model	#heads	d_kv	d_ff
T5-Large	1,024	16	64	4,096
T5-11b	1,024	128	128	∼65k
T5v1.1-xxl	4,096	64	64	∼10k

Table 3 Main differences between used models. d_model is the dimension of the word embeddings, #heads is the number of heads, d_kv is the hidden dimension of the query, key, and value vectors, and d_ff is the dimension of the FF networks after the attention sub-layer.

[3] The pre-trained models can be downloaded from: `https://github.com/google-research/google-research/tree/master/t5_closed_book_qa`

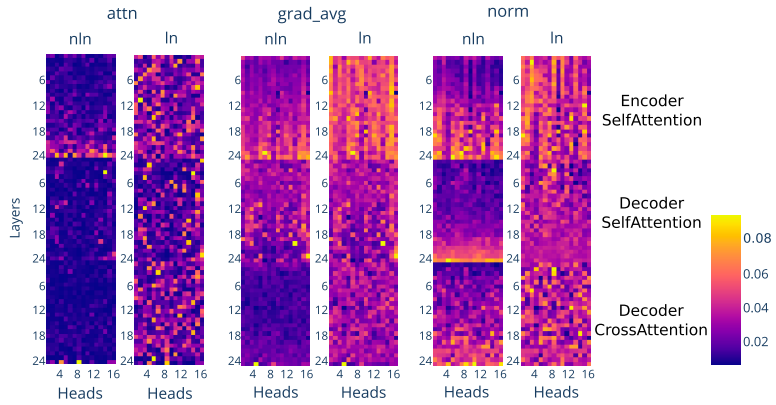

Fig. 4 Head Importance score maps for different scoring methods with T5-large on the NQ dataset. "nln" means no layer normalization and "ln" means layer-normalized.

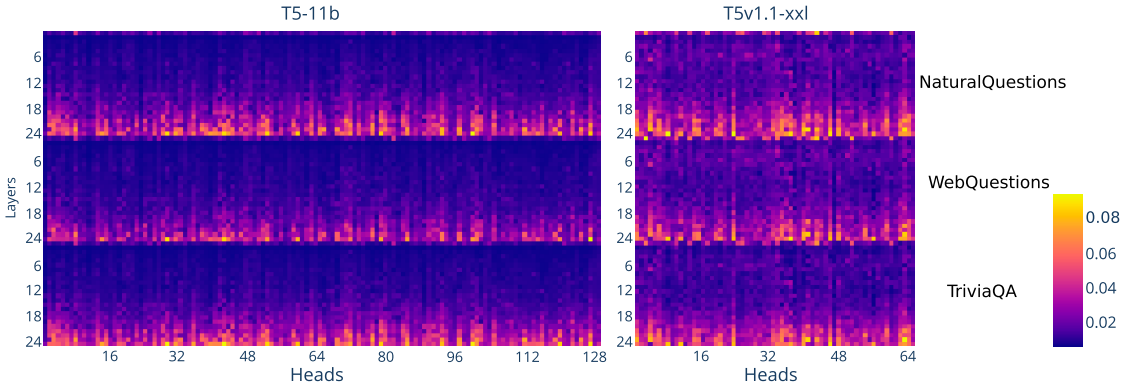

Fig. 5 Comparing head importance visualizations of the encoders of T5-11b and T5v1.1-xxl models between different datasets, using *grad_avg* with "nln"

for speed. Then, we apply the best settings on the huge T5-11b and T5v1.1-xxl variants on the other datasets. We focus on "least-important heads first" in our experiments as it has a more practical benefit.

4.5.1 Least-Important Heads First (LIF)

We prune the least important heads first and do not expect to see a large change in performance. We stop pruning when the EM score reaches below 90% of the original score. In Fig. 6, we show scoring methods compared to the random baseline. The global sorting setting is usually ineffective and achieves below the random baseline; therefore, we ignore it in further experiments. Layer normalization also hurts the importance scores; however, we notice different

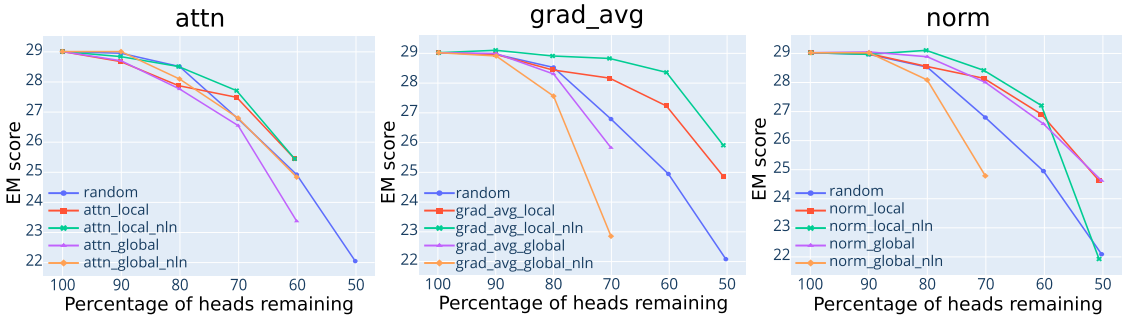

Fig. 6 Comparing pruning experiments with different settings and methods against the random baseline. We compare *attn*, *grad_avg* and *norm* methods introduced in Section 4.2. In the legend, "_local" and "_global" mean local and global sorting explained in Section 4.3.1, while "_nln" means "no layer normalization" as explained in Section 4.3.2.

trends on huge models. We discuss the reasons later. Furthermore, since the *attn* method's performance is closer to the random baseline than other methods, we exclude the *attn* method from comparison. The *attn* method showed relatively reasonable pruning performance in machine translation tasks (Michel et al. 2019) because focusing on which source/target language tokens are attended is essential. Kobayashi et al. (2020) found that only focusing on attention score ignores the norm aspect of vectors.[4]

We run the remaining settings with the huge variants (T5-11b and T5v1.1-xxl) on all datasets and show the results in Fig. 7. While the best pruning method in T5-large is the non-layer-normalized *grad_avg* as shown in Fig. 6, T5v1.1-xxl brings the performance of the *norm* method closer to *grad_avg*. In T5-11b, the *norm* method outperforms the non-layer-normalized *grad_avg* and slightly outperforms the *grad_avg*.

We think the reason is that the standard deviation between examples for the *norm* scores of heads is mostly lower than that for the *grad_avg* method, especially in the areas of low importance. However, small models have a more uniform importance map when normalized, as shown in Fig. 4. Such distribution leads the pruning towards a more random selection process. In models with many heads, layer-normalization produces less uniform maps.

Figs. 6 and 7 show that for *grad_avg*, layer normalization becomes more important as the number of heads in a model increases. We find that non-normalized importance maps tend to assign more importance to later layers as models increase in size, and least important heads just

[4] Experiments of *attn* on other datasets and variants were also conducted, but because of poor performance we exclude it from graphs for visibility.

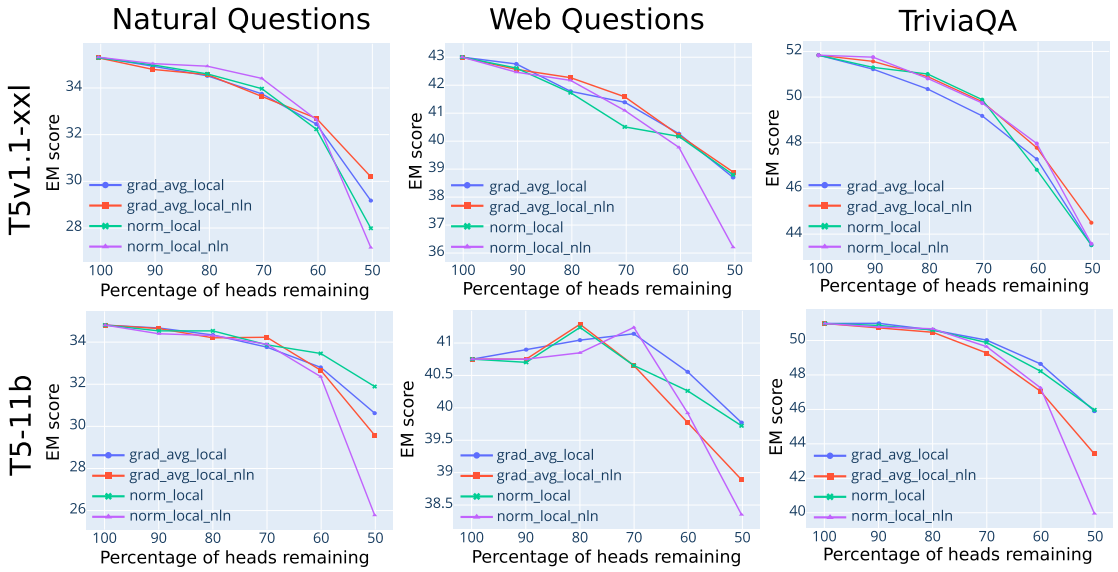

Fig. 7 Least-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using the best performing settings and methods.

become "heads of earlier layers." However, when normalized by layer, the least important heads from all layers are pruned. This is not a problem in small models because when the number of heads is small (e.g., 16), heads of medium importance are not condensed only in later layers. We focus on heads of medium importance here because we notice in Fig. 7 that the sharpest drop between normalized and non-normalized experiments is when 50–60% of heads are remaining.

To understand the intuition of why performance does not decrease much at the beginning of the pruning, we point out that removing the least important heads in the *norm* method is akin to removing the least impactful **z** vectors, as vectors with a small norm value are closer to a zero vector. Intuitively, smaller vectors contribute less than longer ones in reaching the embedded location of an answer when combined. For the *grad_avg* method, the least changed heads mean correcting the vectors in those heads was not necessary to find the correct answer. Therefore, they are probably not crucial for finding the answer.

4.5.2 Most-important Heads First (MIF)

In this mode, we prune important heads first and expect to see a rapid drop in performance. We stop pruning when 50% of heads are pruned. We show a method comparison in Fig. 8 only with "nln" setting, as we find that normalizing by layer loses valuable information about important heads. The *norm* method finds important heads better in T5-large; however, we

find that, although this behavior is consistent between datasets, it differs with model changes. In T5-11b, *grad_avg* and *norm* have similar performance, while in T5v1.1-xxl, *grad_avg* drops faster than *norm*. We show the graphs in Fig. 9. An exception to the pattern of T5-11b is the TriviaQA dataset, where the *norm* method performs worse. We think this is because the size of the evaluation set is over three times that of the other two datasets. This causes the *norm* scores of heads to be averaged more flatly over the 128 heads, with the data distribution being another possible factor.

As the main change in behavior (i.e., *grad_avg* and *norm* methods swapping places in which

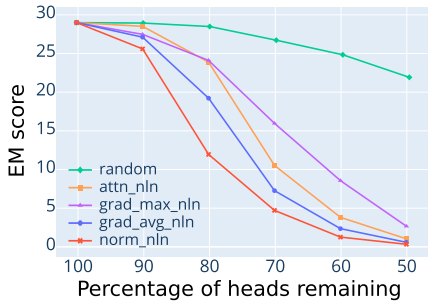

Fig. 8 Pruning most important heads first with different methods using T5-large on the NQ dataset.

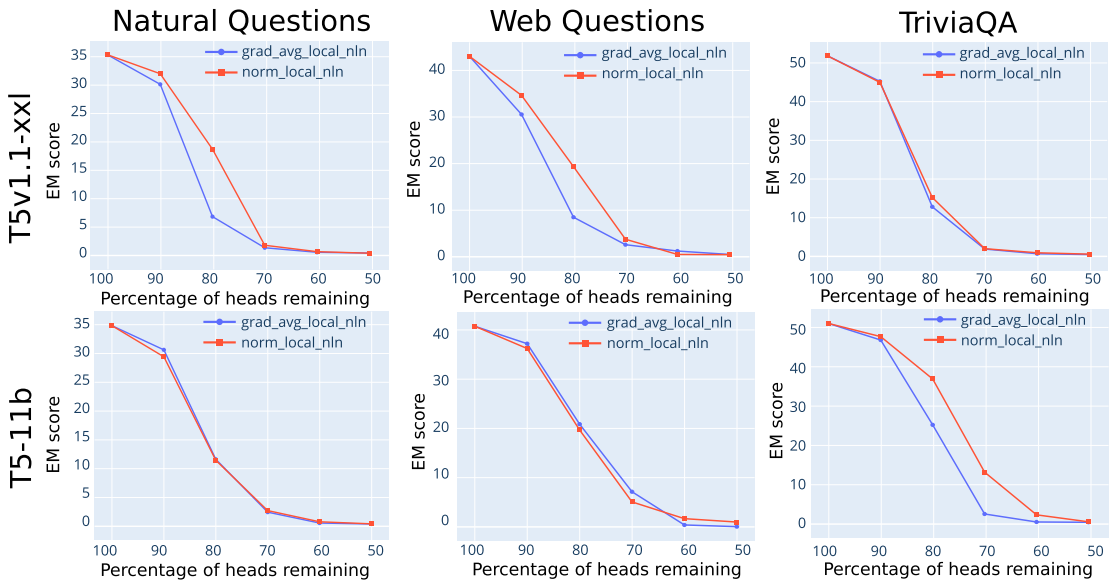

Fig. 9 Most-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using the best-performing settings and methods.

one drops the performance faster) is between T5-large and T5v1.1-xxl, we look at all the differences explained in Section 4.3.3. Then, list the main differences directly influencing the scoring methods between the models and discuss their likelihood of being the responsible change:

- Hidden model size (d_model): T5-large has d_model of $1,024$, while T5v1.1-xxl has $4,096$. While $\mathbf{Z}^h \in \mathbb{R}^{d\_kv}$ did not change between the models, the *grad_avg* was performed on $64 \times 4,096$ instead of $64 \times 1,024$, giving a better hint at what is important because of viewing more gradients, making this change likely responsible.
- The number of heads: T5-large has 16 heads, while T5v1.1-xxl has 64 heads. This increase in heads may not be the reason for the change because T5-11b has 128 heads, yet the two methods perform similarly.
- Feed-forward layers size (d_ff): For T5-large it is $4,096$, while for T5v1.1-xxl it is $10,240$. This change is not responsible for two reasons: T5-11b has a more significant FF layer size of $65,536$ with no drastic behavior change, and the gradients in the FF parameters are negligible, as shown in Fig. 1.

Therefore, we conjecture that the reason for the behavior change is the d_model size. However, many experiments (of changing each factor and observing the performance change) are required to prove all hypotheses. Because their computational costs are unaffordable for a university research lab, this verification is left for future work.

4.5.3 Head Similarity Between Datasets

Fig. 5 showed that models have similar head importance maps across different datasets because of the underlying pre-training phase. We prove this similarity in this section.

We generate three importance maps for each model, one for each dataset (nqo_map, wqo_map, tqao_map) using the *grad_avg* method explained in Section 4.2.3. For example, nqo_map of T5-11b is a head importance map generated by evaluating the T5-11b on the NQ dataset, while the wqo_map is generated with the WQ dataset, and so on. We then apply important-first pruning on a model and a dataset using each of the generated maps (the one generated on the dataset itself and on other datasets as well) and compare the results against a baseline of random head importance, random_map. The results are shown in Fig. 10. The important heads for one dataset are also important for the others because of the shared knowledge in the pre-training phase.

4.6 Distribution of Knowledge per Answer Type

In Section 4.5, we showed that the knowledge stored in T5 is not evenly distributed and that some attention heads store more knowledge than others. However, a question arises: are

there heads that are specialized in certain types of questions? To investigate the distribution for different types of questions (who, where, when, etc.), we analyze the named entity type of the target answer (person, location, date, etc.).

We used SpaCy to do NER on all target answers and we separated the *grad_avg* importance map with "nln" calculated in Section 4.4 by named entity type. The results are shown in Fig. 11.

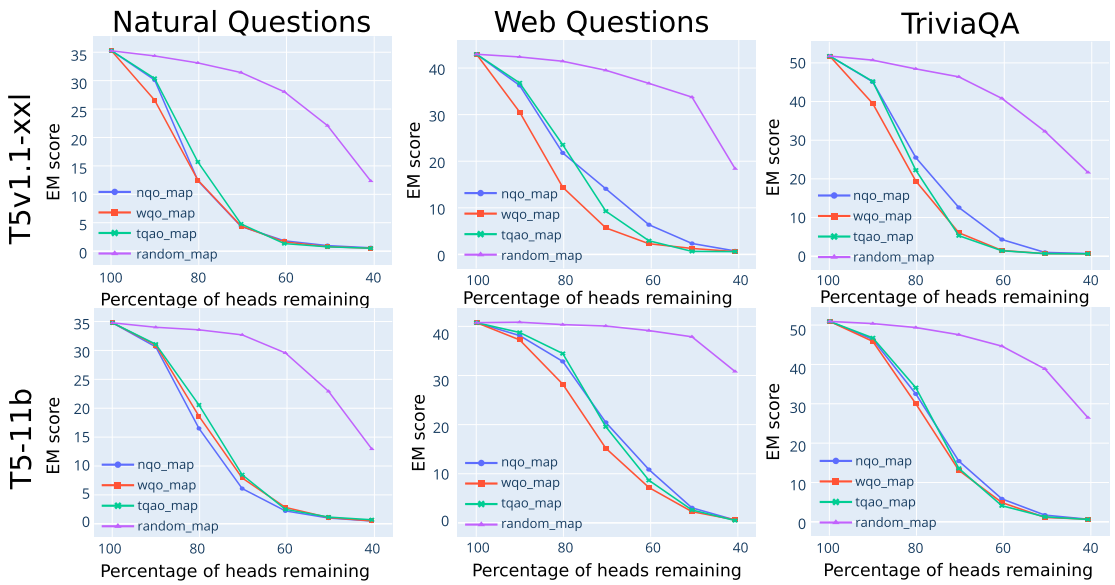

Fig. 10 Most-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using *grad_avg* with "nln" and the importance maps that were calculated on different datasets.

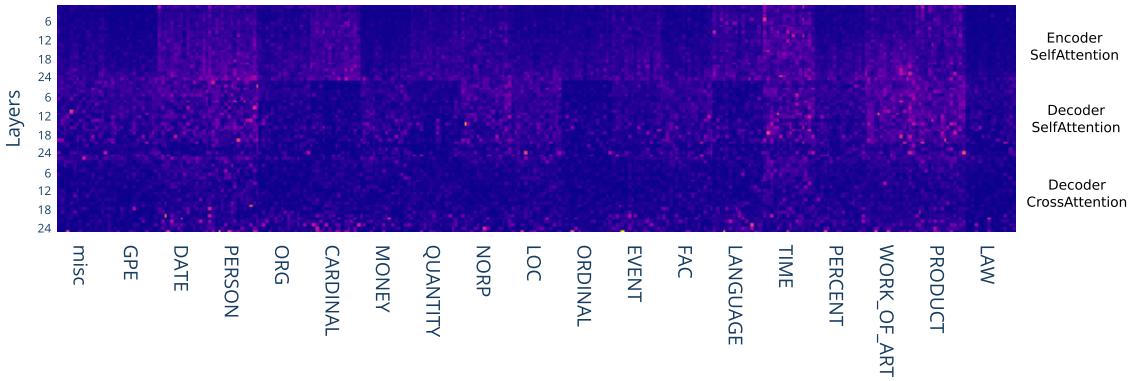

Fig. 11 Distribution of knowledge inside T5-large attention heads depending on the named entity type of the target answer, calculated using the *grad_avg* method.

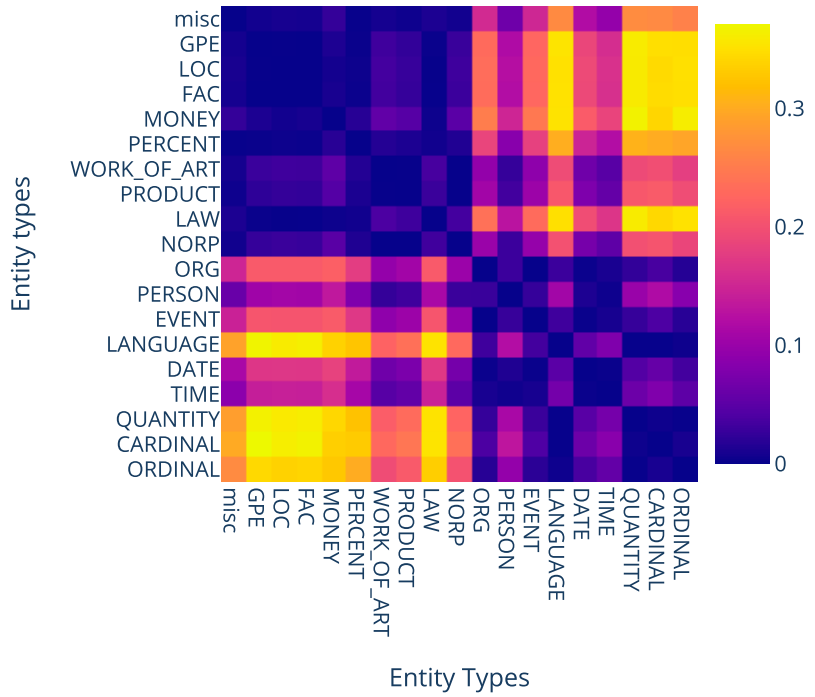

Fig. 12 KL-divergence of importance maps per area between named entity types. Lower divergence represents higher similarity.

Different named entities have different patterns of importance. To determine the similarity of importance maps between entity types, we sum the importance of each type per area (EncSA, DecSA or DecXA).[5] Then, we calculate the Kullback-Leibler (KL) divergence between different entity types and show the results in Fig. 12. We sort the entity types by similarity and notice the correlation with the KL-divergence. In general, similar entity types are stored in similar locations. For example, geopolitical entity (GPE), location (LOC) and facilities (FAC) have similar distribution because they are essentially locations. CARDINAL, QUANTITY, and ORDINAL are similar because they are all related to numbers. DATE and TIME are both related to time.

We note that the results in Section 4 might be dependent on the three experimented datasets. In future work, we want to experiment with other datasets, including ones with different domains.

[5] We calculated divergence per head and layer, but the results were not meaningful, as one head or layer may not hold enough information about many types to deduce similarity.

5 Related Work

5.1 Confirming the Existence of Knowledge

Several works investigating knowledge probing are surveyed by Liu et al. (2021): Petroni et al. (2019) have introduced the LAMA probe that tests the factual and common-sense knowledge in LMs. Using four knowledge sources, they convert the fact triples or question-answer pairs into cloze templates, and they evaluate models on how highly they rank the ground truth token. They found that factual knowledge can be recovered "surprisingly well" from pre-trained LMs. Some work explored continuous template learning instead of discrete prompt template search. Also tested on the LAMA benchmark, Zhong et al. (2021) proposed a model OptiPrompt that optimizes in continuous embedding space to find prompts, rather than being limited to a space of discrete tokens.

Roberts et al. (2020) estimated how much knowledge is stored in LMs by evaluating T5 models on open-domain QA in a closed-book setting. Compared to other competing models, they show that LMs can achieve competitive results on open-domain QA benchmarks without access to context or external knowledge. Different from previous work, we investigate how Transformer-based T5 stores knowledge for closed-book QA.

5.2 Methods of Importance Scoring

Michel et al. (2019) found that a percentage of heads can be removed with no significant impact on the performance. Their criteria for assigning the importance of heads are explained in Section 4.2.2. They effectively give a higher importance score if the attention score is high, which, as presented in Section 4.5, is insufficient. Voita et al. (2019) also focused on the translation task and used several scoring methods: head "confidence" is the average of its maximum attention score over tokens in a sentence. As this method focuses on the "maximum" attention score, it will have the similar problems as the *attn* method in Section 4.2.2. They also used a method called layer-wise relevance propagation (LRP), which has similar behavior to the "confidence" method. They also performed pruning experiments based on stochastic gates and relaxed L_0 penalty criteria. However, their pruning method requires fine-tuning to prune heads, and their results coincide with that of their LRP analysis. Hence, we exclude their methods from our comparison. In this work, we introduce other methods and compare them with the *attn* method.

While we only focus on attention head pruning, Prasanna et al. (2020) experimented with other techniques, such as magnitude pruning, where sparse weights are pruned according to their magnitude and whole layers pruning. However, these techniques are not informative as they are

either quite fine-grained or overly general.

6 Conclusion

This study provided an insight into how Transformer-based T5 operates in a closed-book QA setting. We showed what components of the T5 are important in retrieving the factual knowledge within. We explored the importance of heads in retaining knowledge via pruning while using different settings and scoring methods. In the future, we will investigate other Transformer-based LMs, and how they work on NLP tasks with context, such as open-book QA.

References

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). "Semantic Parsing on Freebase from Question-Answer Pairs." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544.

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). "Reading Wikipedia to Answer Open-domain Questions." *arXiv preprint arXiv:1704.00051*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). "Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding." *arXiv preprint arXiv:1810.04805*.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. (2020). "Realm: Retrieval-augmented Language Model Pre-training." *arXiv preprint arXiv:2002.08909*.

Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. (2020). *spaCy: Industrial-strength Natural Language Processing in Python*. Zenodo.

Jiang, Z., Xu, F. F., Araki, J., and Neubig, G. (2020). "How Can We Know What Language Models Know?" *Transactions of the Association for Computational Linguistics*, **8**, pp. 423–438.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). "Triviaqa: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension." *arXiv preprint arXiv:1705.03551*.

Kobayashi, G., Kuribayashi, T., Yokoi, S., and Inui, K. (2020). "Attention is Not Only a Weight: Analyzing Transformers with Vector Norms." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7057–7075, Online. Association for Computational Linguistics.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., et al. (2019). "Natural Questions: A Benchmark for Ques-

tion Answering Research." *Transactions of the Association for Computational Linguistics*, **7**, pp. 453–466.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing." *arXiv preprint arXiv:2107.13586*.

Michel, P., Levy, O., and Neubig, G. (2019). "Are Sixteen Heads Really Better than One?" *arXiv preprint arXiv:1905.10650*.

Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., and Riedel, S. (2019). "Language Models as Knowledge Bases?" *arXiv preprint arXiv:1909.01066*.

Prasanna, S., Rogers, A., and Rumshisky, A. (2020). "When Bert Plays the Lottery, All Tickets Are Winning." *arXiv preprint arXiv:2005.00561*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). "Exploring the Limits of Transfer Learning with A Unified Text-to-Text Transformer." *arXiv preprint arXiv:1910.10683*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). "Squad: 100,000+ Questions for Machine Comprehension of Text." *arXiv preprint arXiv:1606.05250*.

Roberts, A., Raffel, C., and Shazeer, N. (2020). "How Much Knowledge Can You Pack Into the Parameters of a Language Model?" *arXiv preprint arXiv:2002.08910*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). "Attention Is All You Need." *arXiv preprint arXiv:1706.03762*.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019). "Analyzing Multi-head Self-attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned." *arXiv preprint arXiv:1905.09418*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). "Transformers: State-of-the-Art Natural Language Processing." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online. Association for Computational Linguistics.

Zhong, Z., Friedman, D., and Chen, D. (2021). "Factual Probing Is [MASK]: Learning vs. Learning to Recall." In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5017–5033, Online. Association for Computational Linguistics.

Tareq Alkhaldi: received his B.S. in Information Technology from Arab American University in 2010, and his M.S. in Informatics from Kyoto University in 2018. He is currently working towards a doctoral degree at Kyoto University. His research interests include natural language processing, and in particular, question answering and knowledge representation.

Chenhui Chu: received his B.S. in software engineering from Chongqing University in 2008, and his M.S. and Ph.D. in Informatics from Kyoto University in 2012 and 2015, respectively. He is currently a program-specific associate professor at Kyoto University. His research interests include natural language processing, particularly machine translation and multimodal machine learning.

Sadao Kurohashi: received the B.S., M.S., and Ph.D. in Electrical Engineering from Kyoto University in 1989, 1991 and 1994, respectively. He has been a visiting researcher of IRCS, University of Pennsylvania in 1994. He is currently a professor of the Graduate School of Informatics at Kyoto University. His research interests include natural language processing, knowledge acquisition/representation, and information retrieval. He received the 10th anniversary best paper award from the Journal of Natural Language Processing in 2004, 2009 Funai IT promotion award, and 2009 IBM faculty award.

(Received December 1, 2021)
(Revised February 24, 2022)
(Accepted April 8, 2022)