AI Open 2 (2021) 197–204

Contents lists available at ScienceDirect

AI Open

journal homepage: www.sciencedirect.com/journal/ai-open

Towards a universal continuous knowledge base

Gang Chen [a,c,d], Maosong Sun [a,c,d,e], Yang Liu [a,b,c,d,e,*]

[a] Department of Computer Science and Technology, Tsinghua University, China
[b] Institute for AI Industry Research, Tsinghua University, China
[c] Institute for Artificial Intelligence, Tsinghua University, China
[d] Beijing National Research Center for Information Science and Technology, China
[e] Beijing Academy of Artificial Intelligence, China

ARTICLE INFO

Keywords:
Artificial intelligence
Continuous knowledge base
Neural network

ABSTRACT

In artificial intelligence (AI), knowledge is the information required by an intelligent system to accomplish tasks. While traditional knowledge bases use discrete, symbolic representations, detecting knowledge encoded in the continuous representations learned from data has received increasing attention recently. In this work, we propose a method for building a continuous knowledge base (CKB) that can store knowledge imported from multiple, diverse neural networks. The key idea of our approach is to define an interface for each neural network and cast knowledge transferring as a function simulation problem. Experiments on text classification show promising results: the CKB imports knowledge from a single model and then exports the knowledge to a new model, achieving comparable performance with the original model. More interesting, we import the knowledge from multiple models to the knowledge base, from which the fused knowledge is exported back to a single model, achieving a higher accuracy than the original model. With the CKB, it is also easy to achieve knowledge distillation and transfer learning. Our work opens the door to building a universal continuous knowledge base to collect, store, and organize all continuous knowledge encoded in various neural networks trained for different AI tasks.

1. Introduction

The past two decades have witnessed the rapid progress of deep learning (Hinton and Salakhutdinov, 2006), which has proven effective in learning *continuous representations* from data, leading to substantial improvements in a variety of AI tasks like speech recognition (Dahl et al., 2011), image classification (Krizhevsky et al., 2012), and machine translation (Vaswani et al., 2017). More recently, there has been a significant paradigm shift from learning continuous representations from limited labeled data in a supervised fashion to from abundant unlabeled data in a self-supervised way (Devlin et al., 2019; Brown et al., 2020), which further advances the development of learning continuous representations from data.

Given the remarkable success of deep learning, an interesting question naturally arises: *Is knowledge encoded in the learned continuous representations*? A number of researchers have developed probing methods to evaluate the extent to which continuous representations encode knowledge of interest (Linzen et al., 2016; Belinkov et al., 2017; Blevins et al., 2018; Hewitt and Manning, 2019). For example, while Belinkov et al. (2017) conduct a quantitive evaluation that sheds lights on the ability of neural machine translation models to capture word structure, Hewitt and Manning (2019) propose a structural probe that can test whether syntax trees are consistently embedded in a linear transformation of word representation space.

If knowledge can be defined as the information required by a system to accomplish AI tasks, we would like to distinguish between two categories of knowledge: *discrete* and *continuous*. While discrete knowledge like triples in knowledge graphs uses symbolic representations explicitly handcrafted by humans, continuous knowledge is implicitly encoded in neural networks automatically trained on data. If each trained neural network can be seen as a repository of continuous knowledge, there is an important question that needs to be answered: *Is it possible to build a universal continuous knowledge base that stores knowledge imported from diverse neural networks?*

In this work, we propose a method for building a universal continuous knowledge base (CKB). As shown in Fig. 1, the CKB allows for knowledge transferring between multiple, diverse neural networks. The knowledge encoded in one neural network can be imported to the CKB,

[*] Corresponding author. Department of Computer Science and Technology, Tsinghua University, China.
E-mail address: liuyang2011@tsinghua.edu.cn (Y. Liu).

https://doi.org/10.1016/j.aiopen.2021.11.001
Received 26 October 2021; Accepted 20 November 2021
Available online 24 November 2021

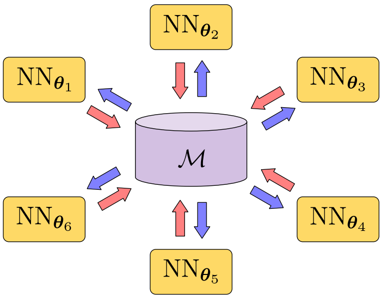

Fig. 1. A universal continuous knowledge base. $\mathcal{M}$ denotes the knowledge base. NN_{θ_n} $(n \in [1,6])$ denotes the n-th neural network parameterized by $\boldsymbol{\theta}_n$. Our goal is use $\mathcal{M}$ to store and accumulate knowledge from various neural networks.

from which the stored knowledge can be exported to another neural network. As a neural network can be seen as a parameterized function, we treat the function and its parameters as the continuous knowledge encoded in the neural network (Section 2.1). Using a memory hierarchy to represent the knowledge base (Section 2.2), our approach defines an interface function for each neural network and casts importing and exporting knowledge as a function simulation problem (Sections 2.3 and 2.5). We adopt multi-task training to import knowledge from multiple neural networks (Section 2.4). Experiments on text classification show that our method is able to fuse knowledge imported from multiple, diverse neural networks and obtain better performance than single neural networks. It is also easy to use the knowledge base to simulate other learning paradigms such as knowledge distillation and transfer learning.

2. Approach

2.1. Knowledge encoded in a neural network

A neural network can be seen as a parameterized, composite function. For example, Fig. 2 shows a simple feed-forward neural network involving the composition of two non-linear functions:

$$\mathbf{h} = f(\mathbf{W}_{xh}\mathbf{x}) \tag{1}$$

$$\mathbf{y} = g(\mathbf{W}_{hy}\mathbf{h}) \tag{2}$$

where $\mathbf{x}$ is the input layer, $\mathbf{h}$ is the hidden layer, $\mathbf{y}$ is the output layer, $\mathbf{W}_{xh}$ is the weight matrix between the input and hidden layers, $\mathbf{W}_{hy}$ is the weight matrix between the hidden and output layers, and $f(\cdot)$ and $g(\cdot)$ are two non-linear functions. For simplicity, we omit bias terms. The neural network shown in Fig. 2 can also be denoted by

$$\mathbf{y} = \text{FFN}_{\boldsymbol{\theta}}(\mathbf{x}) = g_{\theta_2}(f_{\theta_1}(\mathbf{x})) \tag{3}$$

where $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$ is a non-linear function parameterized by $\boldsymbol{\theta} = \{\mathbf{W}_{xh}, \mathbf{W}_{hy}\}$, $f_{\theta_1}(\cdot)$ is a non-linear function parameterized by $\boldsymbol{\theta}_1 = \{\mathbf{W}_{xh}\}$, and $g_{\theta_2}(\cdot)$ is a non-linear function parameterized by $\boldsymbol{\theta}_2 = \{\mathbf{W}_{hy}\}$.

We distinguish between two functions related to neural networks:

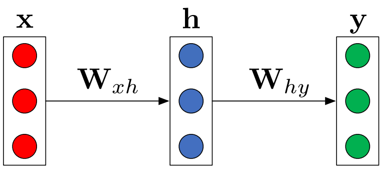

Fig. 2. Example of a feed-forward neural network that can be seen as a composite function. We treat the parameterized function as the continuous knowledge encoded in the neural network. $\mathbf{x}$ is the input, $\mathbf{h}$ is the hidden state, and $\mathbf{y}$ is the output. $\mathbf{W}_{xh}$ and $\mathbf{W}_{hy}$ are parameters.

1. Global function: a function that maps the input of a neural network to its output.
2. Local function: a function that participates in the composition of a global function.

For example, $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$ is a global function and $f_{\theta_1}(\cdot)$ is a local function.

Given a trained neural network that is able to accomplish an AI task, we believe that it is the *parameterized function* that represents the continuous knowledge encoded in the neural network. It is important to note that both the function and the parameters are indispensable. On one hand, if a function has no parameters (e.g., the max pooling function), it can be directly called and there is no need to import it to the knowledge base. On the other hand, as parameters are defined to be bound to a function, parameters themselves are useless if the associated function is missing.

2.2. Continuous knowledge base

2.2.1. Memory hierarchy

To build a CKB, we propose to use two levels of real-valued matrices inspired by the use of memory hierarchy in computer architecture (Hennessy and Patterson, 2011). At the high level, the CKB maintains one real-valued matrix $\mathbf{M}^h$. At the low level, the CKB maintains K real-valued matrices: $\mathbf{M}^l_1,\ldots,\mathbf{M}^l_k,\ldots,\mathbf{M}^l_K$. As a result, the CKB consists of $K + 1$ real-valued matrices:

$$\mathcal{M} = \left\{\mathbf{M}^h, \mathbf{M}^l_1, \ldots, \mathbf{M}^l_K\right\} \tag{4}$$

Note that these real-valued matrices are learnable parameters of the CKB.

2.2.2. Implementation of an interface

To facilitate importing and exporting knowledge between a neural network and the CKB, we introduce an *interface* for the neural network. As shown in Fig. 3, given a neural network $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$, its interface can be defined as a function:

$$\mathbf{y} = \text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}(\mathbf{x}, \mathcal{M}) \tag{5}$$

where $\mathbf{x}$ is the input, $\mathbf{y}$ denotes the output, and $\text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}(\cdot)$ is an interface function parameterized by $\boldsymbol{\varphi}$ tailored for $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$. Note that the input and output of the interface have the same dimensions with those of $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$. With interfaces, we can cast importing and exporting knowledge between the neural network $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$ and the knowledge base as a *function simulation* problem: $\text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}(\mathbf{x}, \mathcal{M})$ runs the same way $\text{FFN}_{\boldsymbol{\theta}}(\mathbf{x})$ does given the same input $\mathbf{x}$ (see Sections 2.3, 2.4, and 2.5 for details).

Fig. 4 shows an example that illustrates how an interface works. Given an input $\mathbf{x}$, our approach first adds two extra matrices on the fly:

$$\widetilde{\mathbf{M}}^l = \mathbf{x}\mathbf{W}^l, \widetilde{\mathbf{M}}^h = \mathbf{x}\mathbf{W}^h \tag{6}$$

where $\mathbf{W}^l$, $\mathbf{W}^h \in \boldsymbol{\varphi}$ are two interface parameters.

Then, for each low-level matrix $\mathbf{M}^l_k$, we use the attention function (Vaswani et al., 2017) to obtain a hidden state:

$$\begin{aligned} \widetilde{\mathbf{M}}^l_k &= \left[\mathbf{M}^l_k; \widetilde{\mathbf{M}}^l\right], \ k = 1, \ldots, K \\ \mathbf{h}_k &= \text{Attention}(\mathbf{x}\mathbf{W}^{\text{q}}, \widetilde{\mathbf{M}}^l_k\mathbf{W}^{\text{k}}, \widetilde{\mathbf{M}}^l_k\mathbf{W}^{\text{v}}) \end{aligned} \tag{7}$$

where $\mathbf{W}^{\text{q}}$, $\mathbf{W}^{\text{k}}$, and $\mathbf{W}^{\text{v}}$ are the transformation matrices for query, key, and value used in the attention mechanism, respectively.

Finally, the output is calculated by calling the attention function again:

$$\mathbf{y} = \text{Attention}(\mathbf{x}\mathbf{W}^{\text{q}}, \mathbf{H}\mathbf{W}^{\text{k}}, \mathbf{H}\mathbf{W}^{\text{v}}) \tag{8}$$

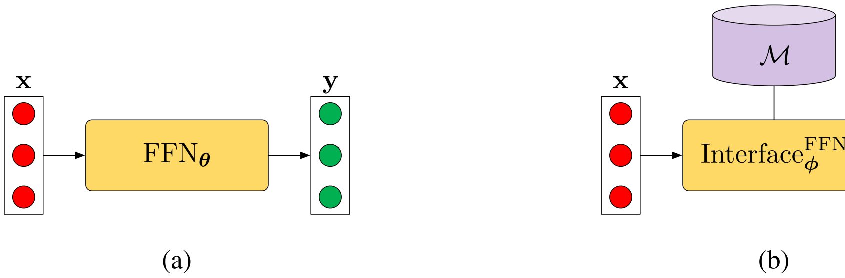

Fig. 3. (a) A feed-forward neural network and (b) its interface to the knowledge base. The neural network is represented as a parameterized function $\mathrm{FFN}_{\boldsymbol{\theta}}(\cdot)$ that takes $\mathbf{x}$ as input and outputs $\mathbf{y}$. Its interface to the knowledge base $\mathcal{M}$ is also defined as a parameterized function $\mathrm{Interface}_{\boldsymbol{\varphi}}^{\mathrm{FFN}}(\cdot)$ that shares the same dimensions of input and output with the neural network. An interface is used to facilitate transferring knowledge between a neural network and the knowledge base.

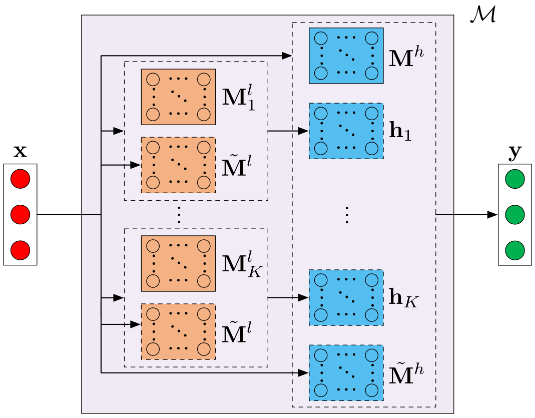

Fig. 4. Illustration of how an interface to the continuous knowledge base works. $\mathcal{M}$ is a continuous knowledge base, which is organized as a memory hierarchy: low-level real-valued matrices $\mathbf{M}_1^l, \ldots \mathbf{M}_K^l$ and a high-level matrix $\mathbf{M}^h$. The knowledge base provides an interface for each neural network. Given an input $\mathbf{x}$, the interface first generates two extra matrices $\widetilde{\mathbf{M}}^l$, $\widetilde{\mathbf{M}}^h$. Note that the matrices generated on the fly are denoted by dashed rectangles. Then, the interface uses the attention function to generate hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_K$, which are concatenated with $\mathbf{M}^h$ and $\widetilde{\mathbf{M}}^h$ to serve as the key and value (i.e., $\mathbf{H}$) of another attention function to generate the output $\mathbf{y}$.

where the hidden state matrix $\mathbf{H}$ is the concatenation of the high-level matrix, the hidden states, and the extra matrix $\widetilde{\mathbf{M}}^h$:

$$\mathbf{H} = \left[\mathbf{M}^h; \mathbf{h}_1; \ldots; \mathbf{h}_K; \widetilde{\mathbf{M}}^h\right] \tag{9}$$

While every neural network has its own interface to the knowledge base and the interface parameters are often different, the continuous knowledge base $\mathcal{M}$ is shared among all neural networks.

2.2.3. *Interfaces for global and local functions*

As the implementation of an interface is transparent to the model structure, it is easy to define an interface for an arbitrary neural network since one only needs to specify the parameterized function and the dimensions of its input and output.

Often, it is convenient to directly define an interface for the global function if there are only a small number of parameters. For example, the global function for the feed-forward neural network shown in Fig. 2 is $\mathrm{FFN}_{\boldsymbol{\theta}}(\cdot)$, we can use Eq. (5) to define its interface.

However, it is more efficient to define an interface for a local function if the neural network calls it frequently. Consider a recurrent neural network defined as a global function:

$$\mathbf{y} = \mathrm{RNN}_{\boldsymbol{\theta}}(\mathbf{x}_{1:T}) \tag{10}$$

where $\mathbf{x}_{1:T} = \mathbf{x}_1, \ldots, \mathbf{x}_T$ is the input sequence and $\mathbf{y}$ is the output.

The global function $\mathrm{RNN}_{\boldsymbol{\theta}}(\cdot)$ runs by calling the local function $f_{\boldsymbol{\theta}}(\cdot)$ repeatedly:

$$\mathbf{h}_t = f_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{h}_{t-1}), \; t = 1, \ldots, T \tag{11}$$

where $\mathbf{x}_t$ and $\mathbf{h}_t$ are the input and the hidden state at time step t, respectively. Note that we let $\mathbf{y} = \mathbf{h}_T$ for simplicity.

As a result, instead of defining an interface for the global function $\mathrm{RNN}_{\boldsymbol{\theta}}(\cdot)$, it is more suitable to define an interface for the local function $f_{\boldsymbol{\theta}}(\cdot)$:

$$\mathbf{h}_t = \mathrm{Interface}_{\boldsymbol{\varphi}}^{f}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathcal{M}) \tag{12}$$

2.3. *Importing knowledge from single neural networks*

As shown in Fig. 5, we cast importing knowledge from a neural network to the knowledge base as a *function simulation* (Jiao et al., 2020) problem: the knowledge base stores the knowledge encoded in a neural network only if the corresponding interface runs in the same way the neural network does. For example, to import the knowledge from the feed-forward neural network shown in Fig. 2 to the knowledge base, we require that the following equation holds for an arbitrary input:

$$\forall \mathbf{x} \in \mathcal{X} : \mathrm{Interface}_{\boldsymbol{\varphi}}^{\mathrm{FFN}}(\mathbf{x}, \mathcal{M}) = \mathrm{FFN}_{\boldsymbol{\theta}}(\mathbf{x}) \tag{13}$$

where $\mathcal{X}$ is a set of all possible inputs.

As a result, the importing process is equivalent to an optimization problem. For example, given a set of inputs $\mathcal{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$, importing $\mathrm{FFN}_{\boldsymbol{\theta}}(\cdot)$ to the knowledge base $\mathcal{M}$ is done by

$$\widehat{\mathcal{M}}, \widehat{\boldsymbol{\varphi}} = \underset{\mathcal{M},\boldsymbol{\varphi}}{argmin}\left\{L_{\mathrm{import}}(\mathcal{D}, \mathcal{M}, \boldsymbol{\varphi})\right\} \tag{14}$$

where the loss function is defined as

$$L_{\mathrm{import}}(\mathcal{D}, \mathcal{M}, \boldsymbol{\varphi}) = \sum_{n=1}^{N} \Delta\left(\mathrm{Interface}_{\boldsymbol{\varphi}}^{\mathrm{FFN}}(\mathbf{x}^{(n)}, \mathcal{M}), \mathrm{FFN}_{\boldsymbol{\theta}}(\mathbf{x}^{(n)})\right) \tag{15}$$

We use $\Delta(\cdot)$ (e.g., a cosine function) to measure the difference between the outputs of two functions.

2.4. *Importing knowledge from multiple neural networks*

To import the knowledge encoded in multiple neural networks to the knowledge base, a natural way is to minimize the importing loss functions of these neural networks jointly. For example, let $\mathcal{D}_1 = \{\mathbf{x}^{(m)}\}_{m=1}^{M}$ be a set of inputs for a feed-forward neural network and $\mathcal{D}_2 = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$ be a set of inputs for a convolutional neural network. Note that the two datasets are independent: the input to the feed-forward neural network can be an image and the input to the convolutional neural network can be a natural language sentence.

The importing loss function of the feed-forward neural network can be defined as

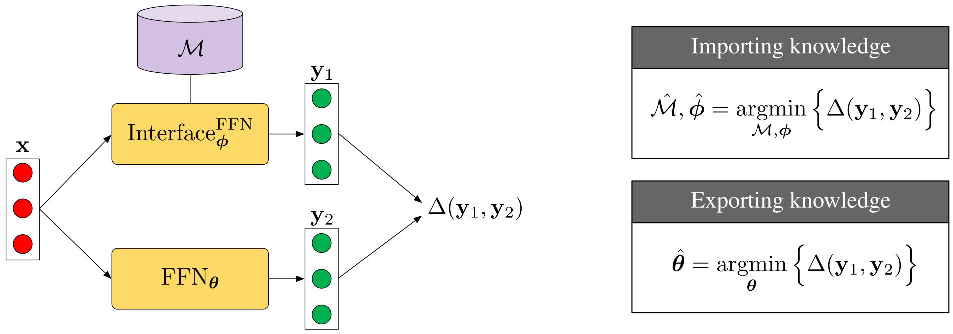
$\mathcal{M}$
$\mathbf{x}$
$\text{Interface}^{\text{FFN}}_{\phi}$
$\text{FFN}_{\boldsymbol{\theta}}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\Delta(\mathbf{y}_1,\mathbf{y}_2)$
Importing knowledge
$\hat{\mathcal{M}}, \hat{\phi} = \underset{\mathcal{M},\phi}{\text{argmin}} \left\{\Delta(\mathbf{y}_1,\mathbf{y}_2)\right\}$
Exporting knowledge
$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \left\{\Delta(\mathbf{y}_1,\mathbf{y}_2)\right\}$

Fig. 5. Importing and exporting knowledge for single neural networks. We cast importing knowledge from a neural network to the knowledge base as a function simulation problem: the knowledge is successfully imported only if the interface $\text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}$ runs the same way the neural network $\text{FFN}_{\boldsymbol{\theta}}$ does. This is done by finding knowledge base and interface parameters (i.e., $\widehat{\mathcal{M}}$ and $\widehat{\boldsymbol{\varphi}}$) that minimize the difference between the outputs of two functions (i.e., $\Delta(\mathbf{y}_1, \mathbf{y}_2)$). Note that the parameters of the neural networks $\boldsymbol{\theta}$ are fixed during importing. Similarly, exporting knowledge is also treated as a function simulation problem: finding model parameters $\widehat{\boldsymbol{\theta}}$ to enable the neural network to imitate the interface while keeping $\mathcal{M}$ and $\boldsymbol{\varphi}$ fixed.

$$L^{\text{FFN}}_{\text{import}}(\mathcal{D}_1,\mathcal{M},\boldsymbol{\varphi}_1) = \sum\nolimits_{m=1}^{M} \Delta\left(\text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}_1}(\mathbf{x}^{(m)},\mathcal{M}), \text{FFN}_{\boldsymbol{\theta}_1}(\mathbf{x}^{(m)})\right) \qquad (16)$$

Similarly, the importing loss function of the convolutional neural network can be defined as

$$L^{\text{CNN}}_{\text{import}}(\mathcal{D}_2,\mathcal{M},\boldsymbol{\varphi}_2) = \sum\nolimits_{n=1}^{N} \Delta\left(\text{Interface}^{\text{CNN}}_{\boldsymbol{\varphi}_2}(\mathbf{x}^{(n)},\mathcal{M}), \text{CNN}_{\boldsymbol{\theta}_2}(\mathbf{x}^{(n)})\right) \qquad (17)$$

Then, importing $\text{FFN}_{\boldsymbol{\theta}_1}(\cdot)$ and $\text{CNN}_{\boldsymbol{\theta}_2}(\cdot)$ to the knowledge base $\mathcal{M}$ synchronously is given by

$$\widehat{\mathcal{M}}, \widehat{\boldsymbol{\varphi}}_1, \widehat{\boldsymbol{\varphi}}_2 = \underset{\mathcal{M},\boldsymbol{\varphi}_1,\boldsymbol{\varphi}_2}{argmin}\left\{L^{\text{FFN}}_{\text{import}}(\mathcal{D}_1,\mathcal{M},\boldsymbol{\varphi}_1) + L^{\text{CNN}}_{\text{import}}(\mathcal{D}_2,\mathcal{M},\boldsymbol{\varphi}_2)\right\} \qquad (18)$$

It is easy to extend the above approach to more than two neural networks.

2.5. *Exporting knowledge to a neural network*

We can also use the interface to export the knowledge stored in the CKB to a neural network. Still, we treat exporting knowledge from the knowledge base to a neural network as a function simulation problem: the knowledge is exported to the neural network only if the neural network runs in the same way the interface does. For example, to export the knowledge from the CKB to the feed-forward neural network shown in Fig. 2, we require that the following equation holds for an arbitrary input:

$$\forall \mathbf{x} \in \mathcal{X} : \text{FFN}_{\boldsymbol{\theta}}(\mathbf{x}) = \text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}(\mathbf{x},\mathcal{M}) \qquad (19)$$

As a result, the importing processing is also equivalent to an optimization problem: our goal is to modify the parameters of the neural network to minimize the difference between the neural network and its interface. For example, given a set of inputs $\mathcal{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$, exporting the knowledge stored in the knowledge base $\mathcal{M}$ to $\text{FFN}_{\boldsymbol{\theta}}(\cdot)$ is done by

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{argmin}\left\{L_{\text{export}}(\mathcal{D},\boldsymbol{\theta})\right\} \qquad (20)$$

where the loss function is defined as

$$L_{\text{export}}(\mathcal{D},\boldsymbol{\theta}) = \sum\nolimits_{n=1}^{N} \Delta\left(\text{Interface}^{\text{FFN}}_{\boldsymbol{\varphi}}(\mathbf{x}^{(n)},\mathcal{M}), \text{FFN}_{\boldsymbol{\theta}}(\mathbf{x}^{(n)})\right) \qquad (21)$$

As the knowledge base and interfaces are fixed during the exporting process, exporting knowledge to multiple neural networks is equivalent to exporting it to single neural networks in parallel.

3. Experiments

We evaluated our approach on text classification on two public datasets: (1) Amazon positive review dataset (Fu et al., 2018), and (2) Yelp polarity review dataset (Zhang et al., 2015). Please refer to Appendix A.1. for more details about these datasets.

We used the following five neural networks tailored for text classification in our experiments:

1. RNN (Liu et al., 2016): recurrent neural network. We used a single gate recurrent unit (GRU) (Cho et al., 2014) layer as the encoder. Its hidden size is set to 256. We defined the interface for RNN at the local level: the input of the interface consists of the t-th word $\mathbf{x}_t$ and the $(t-1)$-th hidden state $\mathbf{h}_{t-1}$ and the output is the t-th hidden state $\mathbf{h}_t$.
2. CNN (Kim, 2014): convolution neural network. We followed Kim (2014) to use three filter windows (i.e., 3, 4, and 5) with 80 feature maps, respectively. We defined its interface at the local level: the input of the interface is a sequence of consecutive words $\mathbf{X}_{t:t+w}$ and the output is a hidden state $\mathbf{h}_t$.
3. ANN (Vaswani et al., 2017): attention-based neural network. We used a single Transformer encoder layer as the encoder. Its hidden size and intermediate size are 256 and 1024, respectively. We defined the interface for ANN at the global level: the input of the interface is the entire sequence $\mathbf{X}$ and the output is the sequence of hidden states $\mathbf{H}$.
4. BERT (Devlin et al., 2019): Bidirectional encoder representations from Transformers. The BERT base model[1] was fine-tuned on the two text classification datasets. We defined the interface for BERT at the global level: the input of the interface is the entire sequence $\mathbf{X}$ and the output is the final output of the BERT encoder. Note that the BERT base model uses 12 attention layers. Our interface directly predicts the output of the 12-th layer.
5. GPT-2 (Radford et al., 2019): a language model based on masked self-attention. We fine-tuned GPT-2[2] for text classification and defined its interface similar to that of BERT.

We removed documents that contain less than 5 words and only retained the first 512 tokens for documents that have more than 512 tokens. For RNN, CNN, and ANN, we used BPE (Sennrich et al., 2016) with 32K operations to preprocess the datasets. For other methods, we used their built-in tokenizers to preprocess the datasets.

We used 10 low-level 30 × 256 matrices and one high-level 20 × 256 matrix to build the CKB for small models (i.e., RNN, CNN, and ANN). For big models like BERT and GPT-2, the CKB contains 20 low-level 40 × 2, 048 metrices and one high-level 20 × 2, 048 matrix. We used AdamW (Loshchilov and Hutter, 2019) to optimize parameters of the CKB, interfaces, and neural networks. Please refer to Appendix A. for more details.

[1] https://huggingface.co/bert-base-uncased.

[2] https://huggingface.co/gpt2.

3.1. *Importing and exporting knowledge for single neural networks*

Table 1 shows the results of importing and exporting knowledge for single neural networks. This experiment aims to verify whether CKB is able to import and export continuous knowledge. We find that our approach is capable of retaining the expressive power of the original neural network across a variety of architectures.

Table 1 also lists the numbers of model, interface, and CKB parameters. The interface for CNN has the fewest parameters (i.e., 0.79M) because it only takes a substring of the input sequence as input. As the interface for RNN takes both the current token and the last hidden state as input, it has more parameters (i.e., 1.05M) than that of CNN. The interface for ANN has more parameters than that of RNN because it takes the entire sequence as input and contains an additional feed-forward layer. Since the interfaces for BERT and GPT-2 directly imitate the output of the final layer (i.e., the 12-th layer), they have much fewer parameters than the original models.

Table 3 shows the results of importing and exporting knowledge for different single models. We import the knowledge from one model to the CKB and then export the knowledge to a different model, which is similar to the setting of zero-shot learning. We find that the knowledge stored in RNN, CNN, and ANN can be transferred to each other with only small performance degradation. As our method uses function simulation to transfer knowledge, it is easy to imitate knowledge distillation and transfer learning based on the CKB (see Appendix B.1. and Appendix B.2.).

3.2. *Importing and exporting knowledge for multiple neural networks*

Table 2 shows the results of importing and exporting knowledge for multiple models on the Amazon dataset. We find that first importing multiple models to the CKB and then exporting the fused knowledge to a single model can result in improved accuracy for the single model. For example, the single model RNN obtains an accuracy of 84.30% while "{RNN, CNN} ↤ CKB ↤RNN" achieves 85.05%, suggesting that the knowledge stored in CKB helps to improve RNN. Similar results were also observed for larger models such as BERT and GPT-2. However, we also found that not all single models to which the knowledge is exported obtain higher accuracies. For example, "{RNN, CNN} ↤ CKB ↤CNN" obtains a lower accuracy than the original CNN. As a result, how to ensure all participating models benefit from the integration still needs further exploration.

Our approach significantly differs from model ensemble in two aspects. First, while model ensemble has to maintain all participating models during inference, CKB can export its knowledge to a single model. Second, model ensemble requires all participating models are trained for the same task while CKB in principle can take advantage of models trained for different tasks. We also conducted experiments to preliminarily probe knowledge importing and exporting for models of different tasks (see Appendix B.3.).

Table 1
Results of importing and exporting knowledge for single models. "RNN ↤ CKB ↤ RNN" denotes first importing the knowledge encoded in the RNN model that achieves an accuracy of 84.30 on the Amazon dataset to CKB, from which the knowledge stored is exported to another RNN model with the same model structure. Note that the number of parameters does not include the word embedding layer and the classification layer.

Configuration	#Param.			Accuracy	
	model ($\boldsymbol{\theta}$)	interface ($\boldsymbol{\varphi}$)	CKB ($\mathcal{M}$)	Amazon	Yelp
RNN	0.20M	–	–	84.30	96.31
RNN ↤ CKB ↤ RNN	–	1.05M	81.92K	84.40	96.32
CNN	0.26M	–	–	84.35	95.95
CNN ↤ CKB ↤ CNN	–	0.79M	81.92K	84.15	95.67
ANN	0.79M	–	–	84.15	93.51
ANN ↤ CKB ↤ ANN	–	1.31M	81.92K	84.15	93.61
BERT	85.64M	–	–	87.90	97.34
BERT ↤ CKB ↤ BERT	–	19.09M	1.70M	87.60	96.69
GPT-2	85.64M	–	–	87.80	97.62
GPT-2 ↤ CKB ↤ GPT-2	–	19.09M	1.70M	87.75	97.13

Table 2
Results of importing and exporting knowledge for multiple models on the Amazon dataset.

Method	Configuration	Accuracy
Single	RNN	84.30
	CNN	84.35
	ANN	84.15
	BERT	87.90
	GPT-2	87.80
Ensemble	RNN & CNN	85.25
	CNN & ANN	85.40
	ANN & RNN	85.00
	RNN & CNN & ANN	85.35
	BERT & GPT-2	88.45
Ours	{RNN, CNN} ↤ CKB ↤ RNN	85.05
	{RNN, CNN} ↤ CKB ↤ CNN	84.05
	{CNN, ANN} ↤ CKB ↤ CNN	84.25
	{CNN, ANN} ↤ CKB ↤ ANN	84.30
	{ANN, RNN} ↤ CKB ↤ ANN	84.35
	{ANN, RNN} ↤ CKB ↤ RNN	85.20
	{RNN, CNN, ANN} ↤ CKB ↤ RNN	84.95
	{RNN, CNN, ANN} ↤ CKB ↤ CNN	83.95
	{RNN, CNN, ANN} ↤ CKB ↤ ANN	84.10
	{BERT, GPT-2} ↤ CKB ↤ BERT	88.20
	{BERT, GPT-2} ↤ CKB ↤ GPT-2	88.35

Table 3
Results of knowledge transferring between different models on the Amazon dataset.

Method	Configuration	Accuracy
Single	RNN	84.30
	CNN	84.35
	ANN	84.15
Ours	RNN ↤ CKB ↤ CNN	83.35
	RNN ↤ CKB ↤ ANN	83.10
	CNN ↤ CKB ↤ RNN	82.70
	CNN ↤ CKB ↤ ANN	83.85
	ANN ↤ CKB ↤ RNN	82.05
	ANN ↤ CKB ↤ CNN	83.10

3.3. *Effect of the capacity of continuous knowledge base*

Table 4 shows the effect of model capacity of CKB on classification accuracy. We find that the accuracy generally rises with the increase of the number of model parameters (i.e., $\mathcal{M}$) and the benefit becomes modest on larger models.

4. Related work

Our work draws inspiration from three lines of research: memory networks, knowledge bases from pre-trained models, and knowledge distillation.

Table 4
Effect of the capacity of the continuous knowledge base on the Amazon dataset.

Configuration	#Param.	Accuracy
BERT	–	87.90
BERT ↤ CKB ↤ BERT	81.92K	86.55
	1.70M	87.60
	3.11M	87.65

4.1. *Memory networks*

Memory networks (MNs) are first proposed by Weston et al. (2015). The proposed MNs reason with inference components combined with a long-term memory which acts as a dynamic knowledge base to store some knowledge from the input. Sukhbaatar et al. (2015) extend MNs to the end-to-end paradigm by introducing the attention mechanism (Bahdanau et al., 2015) to estimate the relevance of each item in the memory. Kumar et al. (2016) propose dynamic memory networks (DMNs) that use episodic memories to help generate better answers to given questions. The episodic memory in DMNs can be updated dynamically according to the input. Nematzadeh et al. (2020) also argue that the separation of computation and storage is necessary and discuss the advantage of improving memory in AI systems. Different from the existing MNs which store the input-related knowledge, the proposed CKB is a global knowledge base that aims to store the knowledge from different competent neural network models.

4.2. *Knowledge bases from pre-trained models*

Recently, a number of works have been studied on what does the pre-trained language model learns (Petroni et al., 2019; Bouraoui et al., 2020; Rogers et al., 2020; Wang et al., 2020). Petroni et al. (2019) convert the fact (i.e., subject-relation-object triple) into the cloze statement to test the factual and commonsense knowledge in the pre-trained language model. By transforming relational triples into masked sentences, Feldman et al. (2019) propose to mine commonsense knowledge from pre-trained models. Bouraoui et al. (2020) fine-tune the pre-trained BERT (Devlin et al., 2019) to predict whether a given word pair is likely to be an instance of some relations. Wang et al. (2020) state that pre-trained language models would be open knowledge graphs and propose an unsupervised method to build knowledge graphs. Rombach and Esser (2020) propose a conditional invertible neural network to translate between fixed representations from different off-the-shelf models. These methods show that neural network models would contain knowledge while our CKB investigates how to store and use this uninterpretable knowledge.

4.3. *Knowledge distillation*

Knowledge distillation (KD) (Hinton et al., 2015) is a common technique that is usually used for model compression. Initial KD methods distill the knowledge from a large deep neural network into a small network by having the student model mimic the final prediction of the teacher model (Hinton et al., 2015; Ba and Caruana, 2014). Many researchers further propose to use the intermediate representations of the teacher model and the relationships between different layers of the teacher model to supervise the training of the student model. Romero et al. (2015) propose to directly match the feature activations of the teacher and the student. Zagoruyko and Komodakis (2017) use the attention map as the supervisor to improve the training of the student model. Jiao et al. (2020) use both the intermediate representation and the attention map to distill the knowledge from a large BERT (Devlin et al., 2019) into a tiny BERT. Yim et al. (2017) propose a flow of solution process, which is defined by the Gram matrix between two layers, as the distilled knowledge. Our work is inspired by the core idea of KD to transfer knowledge using function simulation. The main difference is that we focus on how to store knowledge from multiply, diverse neural networks into a universal continuous knowledge base.

5. Conclusion and future work

We propose to build a universal continuous knowledge base (CKB) in this work. Different from conventional knowledge bases using discrete symbols to represent information, the proposed CKB stores the knowledge in multi-level real-valued matrices. Based on the formalization where a neural network model is a parameterized composite function that maps the input to the output, our CKB imports the knowledge from the neural network model by learning the mapping between the input and the output with the model-dependent interface. Experiments on text classification show that continuous knowledge can be imported and exported between neural networks and the CKB. Our CKB can also mimic knowledge distillation and transfer learning in a novel paradigm.

Our work has only touched the surface of building a universal continuous knowledge base. There are a number of interesting directions awaiting further exploration: a more sophisticated design of memory hierarchy, integrating neural networks trained for different AI tasks, continual learning that can import multiple neural networks asynchronously, visualization and interpretation of the internal workings, and importing and exporting knowledge between discrete and continuous knowledge bases.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank all the anonymous reviewers for their suggestions. This work was supported by the National Key R&D Program of China (No. 2018YFB1005103), National Natural Science Foundation of China (No. 61925601, No. 61761166008), and Huawei Noah's Ark Lab.

Appendix A. Experimental Details

Appendix A.1. Datasets

We conducted experiments on two widely used datasets:

1. Amazon positive-negative review dataset (Fu et al., 2018). The training set contains 796,000 reviews. The test set contains 4,000 reviews. We split the original test set into two parts: 2,000 reviews as the validation set and 2,000 reviews as the test set. On average, each review contains 19 words.
2. Yelp polarity review dataset (Zhang et al., 2015). The training set contains 560,000 reviews. The test set contains 38,000 reviews. We split the original training set into two parts: 550,000 reviews as the training set and 10,000 as the validation set. On average, each review contains 163 words.

Appendix A.2. Hyper-parameter Values

When importing and exporting continuous knowledge between neural networks and the knowledge base, each mini-batch contains 8,192 tokens for the Amazon dataset and 24,576 tokens for the Yelp dataset, respectively. We used the AdamW optimizer (Loshchilov and Hutter, 2019) with $\beta_1 =$

0.9, $\beta_2 = 0.98$, $\varepsilon = 10^{-9}$, and L2 weight decay of 0.01 to optimize parameters. For knowledge importing, we set the learning rate to 2e-4. For knowledge exporting, the learning rate was set to 1e-4.

Appendix A.3. Model Selection

For importing and exporting knowledge for single neural networks, model selection is done by choosing the checkpoint with the highest accuracy on the validation set. When synchronously importing knowledge from multiple neural networks to the continuous knowledge base (CKB), a problem is that these models might not achieve the highest performance on the validation set at the same time. To address this problem, we select the checkpoint as follows:

$$\widehat{c} = \underset{c}{argmax}\left\{ \min_{i\in[1,N]} \{\mathrm{acc}(c, \mathrm{NN}_i)\} \right\} \tag{A.1}$$

where N is the number of neural networks, NN_i is the i-th neural network, c is a checkpoint, and $\mathrm{acc}(\cdot)$ is a function that calculates accuracy on the validation set.

Appendix B. Auxiliary Experiments

Appendix B.1. Knowledge Distillation via Continuous Knowledge Base

Table Table 5

Results of mimicking knowledge distillation (KD) by CKB on the Amazon dataset.

Method		Hidden Size	Accuracy
Base	RNN	256	84.30
	RNN	64	83.30
KD	RNN	64	83.00
Ours	RNN	64	83.00

We can use the CKB to realize the goal of knowledge distillation (KD) (Hinton et al., 2015). As our CKB can export the stored knowledge to a blank model, KD can be done by importing the knowledge from the teacher model to the CKB and then exporting the knowledge to the student model. In our experiments, we used a big RNN with 256 hidden size and a small RNN with 64 hidden size as the teacher and the student models, respectively. As shown in Table B.5Table B.5, KD with our CKB achieved the same performance as the standard KD method. The performances of two KD methods are slightly worse than that of the small model trained on labeled data. One possible reason is that the teacher model contains noise, affecting the performance of the student model.

Appendix B.2. Transfer Learning via Continuous Knowledge Base

Table Table 6

Results of Transfer Learning by CKB on the Amazon dataset. "PT", "FT", and "TL" denote pre-training, fine-tuning, and transfer learning, respectively. "Init" means model initialization.

Method		Setting	Accuracy
PT & FT	ALBERT	Init + FT	79.20
	ALBERT	PT + FT	84.50
Ours	CKB	Init + FT	82.10
	CKB	TL + FT	84.10

It is easy to imitate transfer learning based on our CKB. In our experiments, we used the CKB to mimic the transfer learning where a pre-trained language model (i.e., ALBERT[3] (Lan et al., 2020)) is fine-tuned for text classification. As shown in Table B.6, the ALBERT trained from scratch on the Amazon dataset obtained 79.20 accuracy scores on the test set while the pre-trained ALBERT can obtain 84.50 accuracy scores after fine-tuning. Analogously, the CKB-based model trained from scratch with a randomly initialized CKB on the Amazon dataset achieved 82.10 accuracy scores on the test set. However, the CKB-based model, which imported the knowledge from the pre-trained ALBERT, performed much better. Note that in the knowledge transfer phase, we only used the unlabeled text data from the Amazon dataset to import the knowledge from the pre-trained ALBERT to the CKB.

Appendix B.3. Importing and Exporting Knowledge for Models of Different Tasks

Table Table 7

Results of importing and exporting knowledge for models of different tasks. "2L" means that the ANN model uses two transformer layers as the encoder.

Method	Configuration	Task	Performance
Single	RNN	Text Classification	84.30
	ANN (2L)	POS Tagging	94.73
Ours	RNN ↤ CKB ↤ RNN	Text Classification	84.40

(*continued on next page*)

[3] https://huggingface.co/albert-base-v2.

Table Table 7 (*continued*)

Method	Configuration	Task	Performance
	ANN (2L) ↤ CKB ↤ ANN (2L)	POS Tagging	94.41
	{RNN, ANN (2L) }↤ RNN	Text Classification	84.80
	{RNN, ANN (2L)} ↤ ANN (2L)	POS Tagging	94.63

Table B.7 shows the results of importing and exporting knowledge for models of two different tasks: text classification and part-of-speech (POS) tagging. For text classification task, we used the RNN model and the Amazon dataset. For POS tagging task, we used the ANN model with the encoder of two transformer layers to conduct experiments on the Penn Treebank-3 (PTB) dataset.[4] For the PTB dataset, we follow Collins (2002) to use sections 0-18 as the training set, sections 19-21 as the valid set, and sections 22-24 as the test set. We report the accuracy and the F1 score for text classification and POS tagging tasks, respectively. We find that first importing the RNN for text classification and the ANN (2L) for POS tagging into the CKB and then exporting the fused knowledge to a new RNN can obtain a higher accuracy than the original RNN. However, exporting the fused knowledge to a new ANN (2L) does not boost the performance.

References

Ba, J., Caruana, R., 2014. Do deep nets really need to be deep?. In: Proceedings of NeurIPS, p. 2014.

Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In: Proceedings of ICLR 2015.

Belinkov, Y., Durrani, N., Dalvi, F., Sajjad, H., Glass, J., 2017. What do neural machine translation models learn about morphology?. In: Proceedings of ACL 2017.

Blevins, T., Levy, O., Zettlemoyer, L., 2018. Deep rnns encode soft hierarchical syntax. In: Proceedings of ACL 2018.

Bouraoui, Z., Camacho-Collados, J., Schockaert, S., 2020. Inducing relational knowledge from BERT. In: Proceedings of AAAI 2020.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: Proceedings of NeurIPS 2020, 2020.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Proceedings of EMNLP 2014.

Collins, M., 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In: Proceedings of EMNLP 2002.

Dahl, G.E., Yu, D., Deng, L., Acero, A., 2011. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans. Audio Speech Lang. Process.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2019. Bert: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT 2019.

Feldman, J., Davison, J., Rush, A., 2019. Commonsense knowledge mining from pretrained models. In: Proceedings of EMNLP-IJCNLP 2019.

Fu, Z., Tan, X., Peng, N., Zhao, D., Yan, R., 2018. Style transfer in text: exploration and evaluation. In: Proceedings of AAAI 2018.

Hennessy, J.L., Patterson, D.A., 2011. Computer Architecture: A Quantitative Approach. Morgan Kaufmann.

Hewitt, J., Manning, C., 2019. A structural probe for finding syntax in word representations. In: Proceedings of NAACL-HLT 2019.

Hinton, G., Salakhutdinov, R., 2006. Reducing the dimensionality of data with neural networks. Science.

Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the Knowledge in a Neural Network. arXiv preprint arXiv:1503.02531.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., Liu, Q., 2020. TinyBERT: distilling bert for natural language understanding. In: Proceedings of EMNLP 2020. Findings.

Kim, Y., 2014. Convolutional neural networks for sentence classification. In: Proceedings of EMNLP 2014.

Krizhevsky, A., Sutskever, I., Hinton, G., 2012. Imagenet classification with deep convolutional neural networks. In: Proceedings of NeurIPS, p. 2012.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., Socher, R., 2016. Ask me anything: dynamic memory networks for natural language processing. In: Proceedings of ICML 2016.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R., 2020. ALBERT: a lite bert for self-supervised learning of language representations. In: Proceedings of ICLR 2020.

Linzen, T., Dupoux, E., Goldberg, Y., 2016. Assessing the Ability of Lstms to Learn Syntax-Sensitive Dependencies. Transactions of the Association for Computational Linguistics.

Liu, P., Qiu, X., Huang, X., 2016. Recurrent neural network for text classification with multi-task learning. In: Proceedings of IJCAI 2016.

Loshchilov, I., Hutter, F., 2019. Decoupled weight decay regularization. In: Proceedings of ICLR 2019.

Nematzadeh, A., Ruder, S., Yogatama, D., 2020. On memory in human and artificial language processing systems. In: Proceedings of ICLR 2020 Workshop.

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., Miller, A., 2019. Language models as knowledge bases?. In: Proceedings of EMNLP-IJCNLP 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., 2019. Language Models Are Unsupervised Multitask Learners. OpenAI Blog.

Rogers, A., Kovaleva, O., Rumshisky, A., 2020. A Primer in Bertology: what We Know about How Bert Works. Transactions of the Association for Computational Linguistics.

Rombach, R., Esser, P., 2020. Network-to-network translation with conditional invertible neural networks. In: Proceedings of NeurIPS, p. 2020.

Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y., 2015. Fitnets: hints for thin deep nets. In: Proceedings of ICLR 2015.

Sennrich, R., Haddow, B., Birch, A., 2016. Neural machine translation of rare words with subword units. In: Proceedings of ACL 2016.

Sukhbaatar, S., Weston, J., Fergus, R., et al., 2015. End-to-end memory networks. In: Proceedings of NeurIPS 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need. In: Proceedings of NeurIPS 2017.

Wang, C., Liu, X., Song, D., 2020. Language Models Are Open Knowledge Graphs. arXiv preprint arXiv:2010.11967.

Weston, J., Chopra, S., Bordes, A., 2015. Memeory networks. In: Proceedings of ICLR 2015.

Yim, J., Joo, D., Bae, J., Kim, J., 2017. A gift from knowledge distillation: fast optimization, network minimization and transfer learning. In: Proceedings of CVPR 2017.

Zagoruyko, S., Komodakis, N., 2017. Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. In: Proceedings of ICLR 2016.

Zhang, X., Zhao, J., LeCun, Y., 2015. Character-level convolutional networks for text classification. In: Proceedings of NeurIPS, p. 2015.

[4] https://catalog.ldc.upenn.edu/LDC99T42.