# Survey: Knowledge Graph Assisted Deep Learning Based Question Answering System

**Harsh Peswani, Pushpak Bhattacharyya**
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
{harshpeswani, pb}@cse.iitb.ac.in

## Abstract

Question Answering (QA) is one of the essential downstream tasks in NLP. Question Answering in the maintenance domain requires us to utilize user manuals containing instructions to operate the device safely. It is a challenging task as it requires handling multiple types of questions, such as factoid, non-factoid, confirmation, and list types. Moreover, there is a scarcity of data available for the maintenance domain. To tackle this task, we need a pipeline that contains a question-answering system and an information retrieval module. The information retrieval module retrieves all the relevant paragraphs from the user manual to answer the questions. Question Answering module answers all types of questions using the extracted paragraphs.

## 1 Problem Statement

Question Answering is one of the essential downstream task in NLP. The user asks the question to the Question Answering system, and the goal of the system is to answer the user in the natural language.

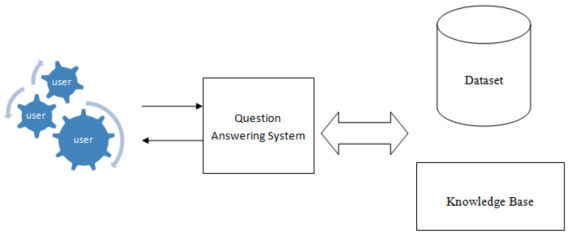


Figure 1: Question Answering (Sultana and Badugu, 2020)

## 2 Challenges and Motivation

User manuals contain instructions to operate the device safely. User manuals typically have 30-40 pages of instructions. Significantly less number of people read the user manual. People usually call technicians when there is a problem with their device, or they have a question about their device. Due to the scarcity of technicians, it takes time for users to get an appointment with the technicians. There is also a huge workload on the technicians. Our solution is simple; users can ask the question-answering system about the problem instead of calling the technicians. The goal of the question-answering system is to give a correct and complete solution to the issue asked by the user, thus reducing the dependency on technicians and the technicians' workload.

There are many challenges while designing such a system. We need an accurate information retrieval system. The answers can be in multiple paragraphs, which may or may not be continuous. The job of the information retrieval module is to find all the paragraphs. The answer will be incomplete if it fails to find all the paragraphs. If it gives a wrong paragraph, the response generated by the question-answering module can be incorrect. Some of the information in the input to the reading comprehension can be noise. Designing such an accurate information retrieval system is a great challenge. User manuals contain images and tables. Inferring answers from them is a great challenge. Transfer learning is an integral part of the system. There is very little data available for question-answering in the maintenance domain. We have to rely on the general domain dataset to solve this problem and ensure it will also work in the maintenance domain.

The second part of the system consists of the question-answering system, which contains even more challenges than the information retrieval system. Questions in user manuals can

be of four types (Reddy and Madhavi, 2017).

- **Factoid Type Questions:** These are fact-based questions. Usually, factoid-type questions start with 'wh-words.' For example, what is the capital of India? The answers are generally named entities.
- **List Type Questions:** These are the questions in which the list of points is the answer. For example, List the name of 10 comedy movies? The answer to this question will be a list of the named entities. Another example can be: List the steps to write a good report? The answer to this will not be a list of the named entities. Instead, it will contain the list of statements in some order.
- **Confirmation Questions [yes or no]:** These are the questions in which the answer is a boolean (yes or no). An example of confirmation-type questions is: Does the sun rises in the east? The answer is yes or no.
- **Non Factoid Type Questions:** These are open-ended questions requiring complex answers. These can be opinions, descriptions, or explanations. An example of non-factoid questions is How to read research papers? The answer to this question will contain some opinions, and the answer will be descriptive.

These four types not only contain challenges of their own but dealing with all four types gives rise to many other challenges. Classifying the question into these four types is difficult, especially between Descriptive and List type questions. It is impossible to classify between descriptive and list types of questions just by looking into the question, as generally, list types of questions are part of descriptive type. This classification depends on how the answer is in the user manual. For instance, if the question is "How to start the dishwasher?" the classification of this question depends on whether the answer in the manual is in points or a paragraph. The questions can be such that the multi-hops are required to answer the question, which is another challenge.

## 3 Literature Survey

Information Retrieval and Question Answering are the tasks on which we focus in this paper. In this section, we will discuss previous works done in this field.

### 3.1 Sentence-BERT

BERT and its variants have excellent results on the sentence-pair tasks (e.g., sentence similarity). Sentences are passed to the transformer network (BERT), which predicts the target value. There is one problem with this approach, assume we have 100 sentences, and the task is sentence similarity. We will have $4950(100 * (100-1)/2)$ pair of sentences. This is no problem for our system. Now, assume instead of 100, we have 10,000 sentences. Now we will have $49995000((10000 * (10000-1)/2)$ pair of sentences, which will take about 65 hours for inference computation. The solution which comes to our mind is to use the sentence embeddings obtained from BERT (averaging or using [CLS] token), but it performs worse than using glove embeddings.

Sentence-BERT uses siamese network architecture to alleviate these problems, which we will see in further sections.

#### 3.1.1 Architecture

Sentence-BERT uses the siamese network to reduce the time complexity at the time of inference. Figure 2 (Reimers and Gurevych, 2019) shows the architecture of the siamese network.

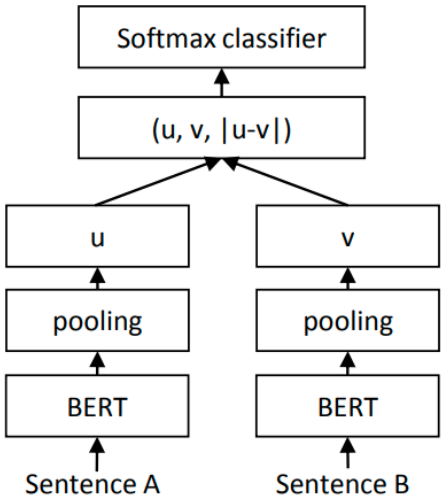


Figure 2: Sentence-BERT Siamese Network

Sentence A and Sentence B are given as an input to the BERT (which shares weights), then there is a pooling layer from which we get embeddings of fixed size length (u and v). u, v, and |u-v| are concatenated and multiplied with the trainable weight. Different objective functions are used for training this network, which we will discuss in the next section.

#### 3.1.2 Training

The architecture of the siamese network is shown in figure 2 (Reimers and Gurevych, 2019). We can see that there is a pooling layer added after BERT to obtain embeddings of the fixed size. The pooling strategy can be MAX pooling, MEAN pooling, or just taking the output of [CLS] token. Different objective functions (Reimers and Gurevych, 2019) which can be used for training the network.

- **Classification Objective Function:** Cross-Entropy loss is generally used for the classification objective. u, v, |u-v| (see figure 3.1.1) are concatenated and multiplied by the trainable weight ($W_t$), then softmax is used for classification.

$$o = softmax(W_t(u, v, |u - v|))$$

- **Regression Objective Function:** Cosine similarity is computed between u and v vectors (see figure 3.1.1), then MSE is used as the objective function.

- **Triplet Objective Function:** Triplet consists of an anchor sentence ($a$), positive sentence ($p$), and negative sentence ($n$). We want the embeddings of the anchor sentence and positive sentence to be as close as possible, and the embeddings of the anchor sentence and negative sentence should be as distant as possible.

$$loss = max(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0)$$

where $s_a$ is the embedding of the anchor sentence, $s_p$ is the embedding of the positive sentence, $s_n$ is the embedding of the negative sentence.

#### 3.1.3 Results

Figure 1 (Reimers and Gurevych, 2019) shows the results of this model on various datasets. We can see from this result that Sentence BERT is giving the state of the art results in almost all the datasets and tasks. It is performing consistently better, meaning sentence BERT gives sentence embeddings with semantic meaning embedded in them.

Table-2 (Reimers and Gurevych, 2019) shows the result on NLI and STSb datasets with three different pooling strategies. We can use MEAN, MAX, or CLS pooling techniques. By default MEAN pooling strategy should be used. We can see the same from the results.

### 3.2 Language Model Pre-training for Dense Passage Retrieval

Dense Retrieval models are very hard to train, it requires a heavily engineered finetuning process. To deal with this problem, the authors used Condenser (see section 3) network to pre-train the LM model. CoCodenser is an extension to Condenser. In CoCodenser contrastive loss is also used to warm up the passage embedding space.

#### 3.2.1 Condenser

The condenser (Gao and Callan, 2021) consists of three types of layers (each consisting of a stack of transformer blocks). Each type of layer serves different purposes. Figure 3 (Gao and Callan, 2021) shows the architecture of the condenser.

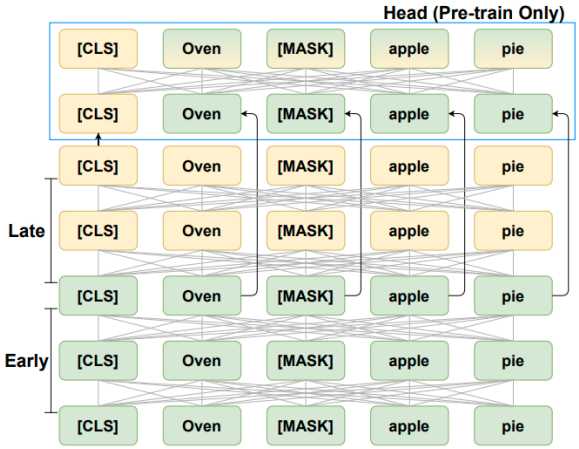


Figure 3: Condenser

First, the input sentence is given to the early layers, then the late layers (see figure (see figure 3 (Gao and Callan, 2021)), here comes the exciting part, input to head layers is given by

| Model | MR | CR | SUBJ | MPQA | SST | TREC | MRPC | Avg. |
|---|---|---|---|---|---|---|---|---|
| Avg. GloVe embeddings | 77.25 | 78.30 | 91.17 | 87.85 | 80.18 | 83.0 | 72.87 | 81.52 |
| Avg. fast-text embeddings | 77.96 | 79.23 | 91.68 | 87.81 | 82.15 | 83.6 | 74.49 | 82.42 |
| Avg. BERT embeddings | 78.66 | 86.25 | 94.37 | 88.66 | 84.40 | 92.8 | 69.45 | 84.94 |
| BERT CLS-vector | 78.68 | 84.85 | 94.21 | 88.23 | 84.13 | 91.4 | 71.13 | 84.66 |
| InferSent - GloVe | 81.57 | 86.54 | 92.50 | **90.38** | 84.18 | 88.2 | 75.77 | 85.59 |
| Universal Sentence Encoder | 80.09 | 85.19 | 93.98 | 86.70 | 86.38 | **93.2** | 70.14 | 85.10 |
| SBERT-NLI-base | 83.64 | 89.43 | 94.39 | 89.86 | 88.96 | 89.6 | **76.00** | 87.41 |
| SBERT-NLI-large | **84.88** | **90.07** | **94.52** | 90.33 | **90.66** | 87.4 | 75.94 | **87.69** |

Table 1: Results

| | NLI | STSb |
|---|---|---|
| Pooling Strategy | | |
| MEAN | **80.78** | **87.44** |
| MAX | 79.07 | 69.92 |
| CLS | 79.80 | 86.62 |

Table 2: Results (Classification Objective)

the late and early layers. Late layers give input to the CLS token, and the early layers give input to all other tokens. If the model wants to use the late layers, it is forced to utilize the CLS token. This way, the CLS token will be trained to generate meaningful embeddings. The objective function used to train this network is the MLM objective function.

$$L^{mlm} = \sum_{i \in masked} CrossEntropy(W h_i^{cd}, x_i)$$

#### 3.2.2 coCondenser

coCondenser (Gao and Callan, 2021) uses condenser architecture (see figure 3). coCondenser uses MLM with contrastive loss. While CLS token in condenser network contains sentence embedding, inner product/dot product between pair of sentences with same semantic meaning will not be high because condenser network is not trained in a way that it should give similar embeddings to the sentences with same semantic meaning. coCondenser eliminates this problem by introducing a contrastive learning objective. The contrast learning objective ensures that similar sentences have a high inner product, and sentences with different semantic meanings should have an inner product near zero.

$$L_{ij}^{co} = -\log \frac{\exp(< h_{i1}, h_{i2} >)}{\sum_{k=1}^{n} \sum_{l=1}^{2} I_{ij!=kl} \exp(< h_{ij}, h_{kl} >)}$$

$$L = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{2} [L_{ij}^{mlm} + L_{ij}^{co}]$$

### 3.3 Results

The datasets used for experiments are MS-MARCO, Natural Question Test, and Trivia QA Test.

| Method | MS-MARCO Dev | | Natural Question Test | | | Trivia QA Test | | |
|---|---|---|---|---|---|---|---|---|
| | MRR@10 | R@1000 | R@5 | R@20 | R@100 | R@5 | R@20 | R@100 |
| BM25 | 18.7 | 85.7 | - | 59.1 | 73.7 | - | 66.9 | 76.7 |
| DeepCT | 24.3 | 90.9 | - | - | - | - | - | - |
| docT5query | 27.7 | 94.7 | - | - | - | - | - | - |
| GAR | - | - | 60.9 | 74.4 | 85.3 | 73.1 | 80.4 | 85.7 |
| DPR | - | - | - | 74.4 | 85.3 | - | 79.3 | 84.9 |
| ANCE | 33.0 | 95.9 | - | 81.9 | 87.5 | - | 80.3 | 85.3 |
| ME-BERT | 33.8 | - | - | - | - | - | - | - |
| RocketQA | 37.0 | 97.9 | 74.0 | 82.7 | 88.5 | - | - | - |
| Condenser | 36.6 | 97.4 | - | 83.2 | 88.4 | - | 81.9 | 86.2 |
| DPR-PAQ | | | | | | | | |
| - $BERT_{base}$ | 31.4 | - | 74.5 | 83.7 | 88.6 | - | - | - |
| - $BERT_{large}$ | 31.1 | - | 75.3 | 84.4 | 88.9 | - | - | - |
| - $RoBERTa_{base}$ | 32.3 | - | 74.2 | 84.0 | **89.2** | - | - | - |
| - $RoBERTa_{large}$ | 34.0 | - | **76.9** | **84.7** | **89.2** | - | - | - |
| coCondenser | **38.2** | **98.4** | **75.8** | **84.3** | 89.0 | **76.8** | **83.2** | **87.3** |

Figure 4: Results

Figure 4 (Gao and Callan, 2021) shows the results of the experiments. The metrics used are MRR@10 (mean reciprocal rank) (higher the better) and R@k (higher the better). We can see from the result that the coCondenser performs better than the condenser network by a significant margin. It is due to the contrastive loss. coCondenser is performing consistently better on all three datasets, which shows that coCondenser gives sentence embeddings with embedded semantic meanings.

### 3.4 UnifiedQA

Question Answering has different formats, such as Extractive span selection, multiple-choice, Abstractive, and Confirmation type.

For all these different formats, different models are designed, often with different architectures. These boundaries between models should not exist. Human beings do not think this way.

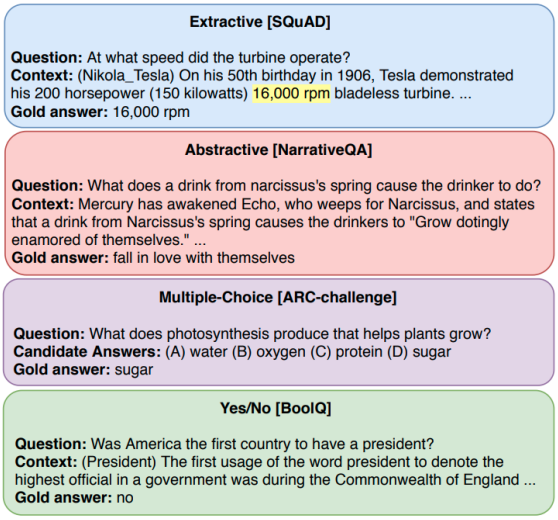


Figure 5: Examples of different formats of Questions

The authors made a single model and showed that such boundaries are not required. The unified model performs equal to or better than the models designed specifically for handling a particular type of format.

UnifiedQA is a single model which can answer all types of questions. The architecture is the same for all formats of questions. Figure 5 (Khashabi et al., 2020) shows an example of questions of each format, i.e., Extractive span selection, multiple-choice, Abstractive, and Confirmation type.

### 3.4.1 Datasets

Datasets used for the training are from all four types of formats of question answering. 5 extractive datasets, 3 abstractive datasets, 9 multiple-choice datasets, and 3 yes/no datasets are used for the experiments.

Figure 6 (Khashabi et al., 2020) shows all 20 datasets which are used for training. Some of the datasets have paragraphs, while some do not have paragraphs. For example, ARC does not have paragraphs. SQUAD 2.0 (Rajpurkar et al., 2018) has idk-type questions, which means that some question cannot be answered by the given paragraph, while all other does not contain idk-type questions.

| Dataset | Train set size | Eval. set size | Best published | 95% CI (%) | Input length | Output length |
|---|---|---|---|---|---|---|
| SQuAD 1.1 | 87$k$ | 10$k$ | 95.6 | 0.4 | 136.2 | 3.0 |
| SQuAD 2.0 | 130$k$ | 11$k$ | 91.2 | 0.5 | 139.9 | 2.6 |
| NewsQA | 76$k$ | 4$k$ | 66.8 | 1.4 | 606.6 | 4.0 |
| Quoref | 22$k$ | 2$k$ | 86.1 | 1.5 | 352.7 | 1.7 |
| Quoref-CS | - | 700 | 55.4 | 3.6 | 324.1 | 2.2 |
| ROPES | 10$k$ | 1.4$k$ | 61.1 | 2.5 | 169.1 | 1.4 |
| ROPES-CS | - | 974 | 32.5 | 3.0 | 182.7 | 1.3 |
| NarQA | 65$k$ | 21$k$ | 58.9 | 0.7 | 563.6 | 6.2 |
| NatQA | 79$k$ | 3.6$k$ | 42.2 | 1.6 | 607.0 | 2.2 |
| DROP | 77$k$ | 9$k$ | 89.1 | 0.6 | 189.1 | 1.6 |
| DROP-CS | - | 947 | 54.2 | 3.2 | 206.0 | 2.1 |
| RACE | 87$k$ | 4$k$ | 89.5 | 0.9 | 317.9 | 6.9 |
| OBQA | 4$k$ | 501 | 80.0 | 3.3 | 28.7 | 3.6 |
| MCTest | 1.4$k$ | 320 | 86.5 | 3.4 | 245.4 | 4.0 |
| ARC (easy) | 2$k$ | 2$k$ | 80.0 | 1.7 | 39.4 | 3.7 |
| ARC (chal.) | 1$k$ | 1$k$ | 67.8 | 2.9 | 47.4 | 5.0 |
| CQA | 9.7$k$ | 1.2$k$ | 79.1 | 2.2 | 26.8 | 1.5 |
| WG | 40.3$k$ | 1.7$k$ | 67.5 | 2.2 | 25.2 | 3.0 |
| PIQA | 16.1$k$ | 3$k$ | 79.4 | 1.4 | 49.6 | 20.2 |
| SIQA | 33.4$k$ | 2.2$k$ | 78.0 | 1.7 | 37.3 | 4.7 |
| BoolQ | 9$k$ | 3$k$ | 91.0 | 1.0 | 105.1 | 1.0 |
| BoolQ-CS | - | 461 | 71.1 | 4.0 | 108.9 | 1.0 |
| NP-BoolQ | 10$k$ | 3$k$ | 78.4 | 1.4 | 106.2 | 1.0 |
| MultiRC | - | 312 | 91.7 | 2.6 | 293.3 | 1.0 |

Figure 7: Statistics of the Datasets

Figure 7 (Khashabi et al., 2020) shows the statistics of the datasets. The table also shows the input length, output length, and best-published result on that particular dataset. If we combine all the datasets, 1.5 Million Questions are used to train the model. Average answer lengths are from 1 word to 20.2 words.

### 3.4.2 Encoding

Datasets that will be used for training are of different types. Some contain paragraphs, while some do not. Some dataset types are MCQ types, while others are not. Naturally, the question arises of how we can give input to any model as there are different types of datasets. There must be the same or similar way to give input to the model. The authors proposed that all datasets will be encoded in one particular format. The format is question options (if present) paragraph (if present)). All the data points present in the dataset will be encoded in this particular format.

| Datasets | SQuAD11 | SQuAD2 | NewsQA | Quoref | ROPES | NarQA | DROP | NatQA | RACE | MCTest | OBQA | ARC | QASC | CQA | WG | PIQA | SIQA | BoolQ | NP-BoolQ | MultiRC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Format | Extractive QA (EX) | | | | | Abstractive QA (AB) | | | Multiple-choice QA (MC) | | | | | | | | | Yes/NO QA (YN) | | |
| Has paragraphs? | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | | | | ✓ | ✓ | ✓ |
| Has explicit candidate ans? | | | | | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | |
| # of explicit candidates | | | | | | | | | 4 | 4 | 4 | 4 | 8 | 5 | 2 | 2 | 3 | | | |
| Para contains ans as substring? | ✓ | ✓ | ✓ | ✓ | | | | | | | | | | | | | | | | |
| Has idk questions? | | ✓ | | | | | | | | | | | | | | | | | | |

Figure 6: Datasets

| | | |
|---|---|---|
| EX | **Dataset** | SQuAD 1.1 |
| | **Input** | `At what speed did the turbine operate? \n (Nikola_Tesla) On his 50th birthday in 1906, Tesla demonstrated his 200 horsepower (150 kilowatts) 16,000 rpm bladeless turbine. ...` |
| | **Output** | `16,000 rpm` |
| AB | **Dataset** | NarrativeQA |
| | **Input** | `What does a drink from narcissus's spring cause the drinker to do? \n Mercury has awakened Echo, who weeps for Narcissus, and states that a drink from Narcissus's spring causes the drinkers to ``Grow dotingly enamored of themselves.'' ...` |
| | **Output** | `fall in love with themselves` |
| MC | **Dataset** | ARC-challenge |
| | **Input** | `What does photosynthesis produce that helps plants grow? \n (A) water (B) oxygen (C) protein (D) sugar` |
| | **Output** | `sugar` |
| | **Dataset** | MCTest |
| | **Input** | `Who was Billy? \n (A) The skinny kid (B) A teacher (C) A little kid (D) The big kid \n Billy was like a king on the school yard. A king without a queen. He was the biggest kid in our grade, so he made all the rules during recess. ...` |
| | **Output** | `The big kid` |
| YN | **Dataset** | BoolQ |
| | **Input** | `Was America the first country to have a president? \n (President) The first usage of the word president to denote the highest official in a government was during the Commonwealth of England ...` |
| | **Output** | `no` |

Figure 8: Text-to-Text Encoding

Figure 8 (Khashabi et al., 2020) shows the example of encoding for all varieties of datasets. Notice that the encoding is the same for all types of datasets. Now we can give any question from the dataset as an input to the model.

### 3.4.3 Model

The model used is T5 (Raffel et al., 2020) (T5 for conditional generation). The input to the model is the encoding which is described in the previous section. The output is a word or a sentence. Figure 9 (Khashabi et al., 2020) shows the pre-training and finetuning of the T5 model.

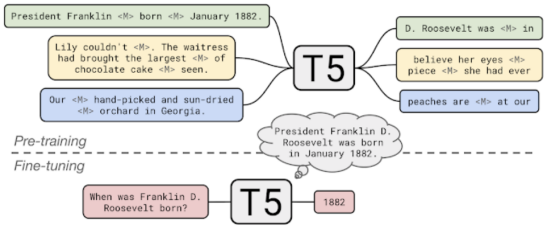


Figure 9: T5

### 3.4.4 Results

The author hypothesizes that the boundaries between different formats are unnecessary; in fact, datasets of one format can help the model answer the question of different formats.

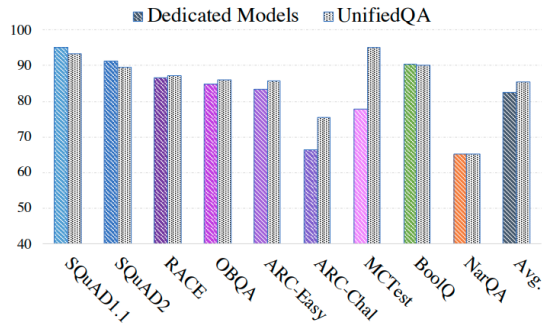


Figure 10: UnifiedQA Results

Figure 10 (Khashabi et al., 2020) shows the same. Unified QA, on average, performs better than all the dedicated models combined, showing that datasets of one format can help the model answer the question of a different format.

## 3.5 Dense passage retrieval for open-domain question answering

This section will discuss another technique that uses dense vectors to retrieve the relevant paragraphs. Similar to SentenceBERT, this technique also generates embedding for questions and paragraphs. The major difference is that the same model generates the embedding of question and paragraph in the case of sentenceBERT. In contrast, in the case of DPR

| Dataset | Train | | Dev | Test |
|---|---|---|---|---|
| Natural Questions | 79,168 | 58,880 | 8,757 | 3,610 |
| TriviaQA | 78,785 | 60,413 | 8,837 | 11,313 |
| WebQuestions | 3,417 | 2,474 | 361 | 2,032 |
| CuratedTREC | 1,353 | 1,125 | 133 | 694 |
| SQuAD | 78,713 | 70,096 | 8,886 | 10,570 |

Table 3: Number of questions in the Datasets

(Karpukhin et al., 2020), a different model generates the embedding for the question and passages.

#### 3.5.1 Training

The authors finetuned two independent BERT to get the embedding of questions and passages. The loss function they used ensures that embedding of question and relevant passage is similar and embedding of the question with other passages is not similar. Naturally, we need a training dataset that contains questions paired with relevant passages for positive examples and questions paired with irrelevant passages for negative examples.

**Negative Examples**

To generate negative examples, the authors used random sampling from positive passages paired with other questions which appear in the training set. Using this technique, we can get negative examples and train the network.

#### 3.5.2 Dataset

The authors used various datasets to train the network. The datasets used for training are shown in table-3 (Karpukhin et al., 2020). The figure shows the number of question passage pairs in the train, dev, and test set.

#### 3.5.3 Results

Metrics used for comparing different models are top-20 and top-100. Top-20 accuracy is the percentage of the top 20 retrieved passages containing the answer. Top-100 accuracy means the percentage of top 100 retrieved passages that contain the answer.

Table-4 (Karpukhin et al., 2020) shows the result of the DPR. We can see from the table that DPR gives state-of-the-art in all five datasets.

### 3.6 TAPAS: Weakly Supervised Table Parsing via Pre-training

TAPAS (Herzig et al., 2020) uses BERT (Devlin et al., 2018) like architecture. The table is flattened into words, then converted into tokens. The input to the model is a "question [SEP] flattened table". The authors added two classification layers, one for selecting cells and the other for selecting the aggregation operator. Figure-11 (Herzig et al., 2020) shows the

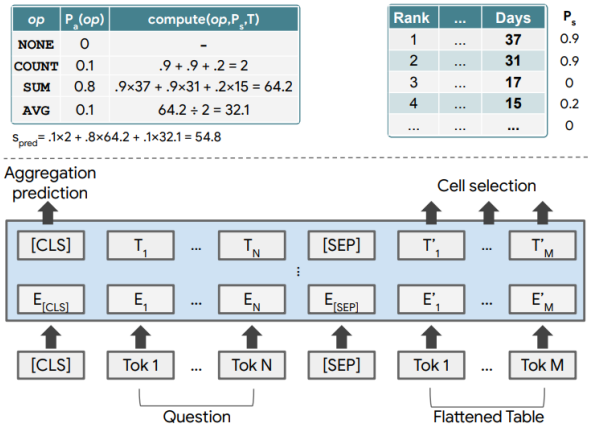


Figure 11: TAPAS Architecture

architecture of the model. We can see that one classification layer selects the aggregation operator and the other classification layers select the subset of table cells.

#### 3.6.1 Embedding of the Table

Along with the token embedding few more things are given as an input to the model. Positional encoding is the same as in BERT. Segment embedding can take two different values, 0 and 1. It takes value 0 if the token is of query, and it takes value one if the token is of the table. Column embedding contains the column number of the cell in the table. Similarly, the row number contains the cell's row number in the table. Figure-12 (Herzig et al.,

| Training | Retriever | Top-20 | | | | | Top-100 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | NQ | TriviaQA | WQ | TREC | SQuAD | NQ | TriviaQA | WQ | TREC |
| None | BM25 | 59.1 | 66.9 | 55.0 | 70.9 | 68.8 | 73.7 | 76.7 | 71.1 | 84.1 |
| 2* Single | DPR | 78.4 | 79.4 | 73.2 | 79.8 | 63.2 | 85.4 | **85.0** | 81.4 | 89.1 |
| | BM25+DPR | 76.6 | 79.8 | 71.0 | 85.2 | **71.5** | 83.8 | 84.5 | 80.5 | 92.7 |
| 2* Multi | DPR | **79.4** | 78.8 | **75.0** | **89.1** | 51.6 | **86.0** | 84.7 | **82.9** | 93.9 |
| | BM25 + DPR | 78.0 | **79.9** | 74.7 | 88.5 | 66.2 | 83.9 | 84.4 | 82.3 | **94.1** |

Table 4: Results

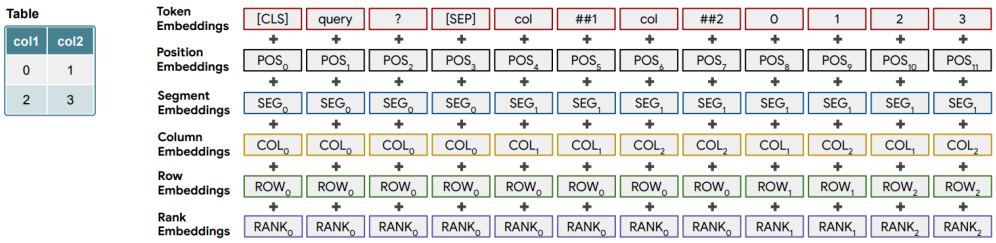


Figure 12: Table Embedding

2020) shows an example of the table embedding in the same figure.

### 3.6.2 Pre-training

Authors mined text and tables from Wikipedia to pre-train the model. Pre-training objective is similar to the BERT, which is masked language modeling. The algorithm masks some of the table's cells, and the end task is to predict those masked tokens. This way, the model can learn interesting correlations between the task and the table's cells.

### 3.6.3 Finetuning

There can be two cases, first is that the answer to the question is directly a subset of cells, or the answer to the question is some aggregator operator on those subsets of cells.

**Case1: Cell Selection**

For cell selection, the loss is given as:

$$\mathcal{J}_{\text{columns}} = \frac{1}{|\text{Columns}|} \sum_{\text{co} \in \text{Columns}} \text{CE}(p_{\text{col}}^{(\text{co})}, \mathbb{1}_{\text{co}=\text{col}})$$

$$\mathcal{J}_{\text{cells}} = \frac{1}{|\text{Cells(col)}|} \sum_{c \in \text{Cells(col)}} \text{CE}(p_{\text{s}}^{(c)}, \mathbb{1}_{c \in C})$$

$$\mathcal{J}_{\text{aggr}} = -\log p_{\text{a}}(op_0)$$

| | WIKISQL | WIKITQ | SQA |
|---|---|---|---|
| Logical Form | | ***X*** | ***X*** |
| Conversational | ***X*** | ***X*** | |
| Aggregation | | | ***X*** |
| Examples | 80654 | 22033 | 17553 |
| Tables | 24241 | 2108 | 982 |

Table 5: Datasets Statistics

Final loss is a linear combination of all three losses.

### 3.6.4 Case2: Scaler Answer

In this case, the answer is a number that may or may not appear in the table. Instead, it is the aggregator of the subset of cells.

$$s_{\text{pred}} = \sum_{i=1} \hat{p}_{\text{a}}(op_i) \cdot \text{compute}(op_i, p_{\text{s}}, T)$$

Where,

$$\hat{p}_{\text{a}}(op_i) = \frac{p_{\text{a}}(op_i)}{\sum_{i=1} p_{\text{a}}(op_i)}$$

| Model | ALL | SEQ | **Q1** | **Q2** | **Q3** |
|---|---|---|---|---|---|
| Pasupat and Liang (2015) | 33.2 | 7.7 | 51.4 | 22.2 | 22.3 |
| Neelakantan et al. (2017) | 40.2 | 11.8 | 60.0 | 35.9 | 25.5 |
| Iyyer et al. (2017) | 44.7 | 12.8 | 70.4 | 41.1 | 23.6 |
| Sun et al. (2018) | 45.6 | 13.2 | 70.3 | 42.6 | 24.8 |
| Müller et al. (2019) | 55.1 | 28.1 | 67.2 | 52.7 | 46.8 |
| TAPAS | **67.2** | **40.4** | **78.2** | **66.0** | **59.7** |

Table 6: Result on SQA dataset

| Model | Test [T 2] |
|---|---|
| Pasupat and Liang (2015) | 37.1 |
| Neelakantan et al. (2017) | 34.2 |
| Haug et al. (2018) | 34.8 |
| Zhang et al. (2017) | 43.7 |
| Liang et al. (2018) | 43.1 |
| Dasigi et al. (2019) | 43.9 |
| Agarwal et al. (2019) | 44.1 |
| Wang et al. (2019) | 44.5 |
| TAPAS | 42.6 |
| TAPAS (pre-trained on WIKISQL) | 48.7 |
| TAPAS (pre-trained on SQA) | 48.8 |

Table 7: Result on WIKITQ dataset

| $op$ | $\text{compute}(op, p_s, T)$ |
|---|---|
| COUNT | $\sum_{c \in T} p_s^{(c)}$ |
| SUM | $\sum_{c \in T} p_s^{(c)} \cdot T[c]$ |
| AVERAGE | $\frac{\text{compute}(\text{SUM}, p_s, T)}{\text{compute}(\text{COUNT}, p_s, T)}$ |

Now, loss is given by:

$$\mathcal{J}_{\text{scalar}} = \begin{cases} 0.5 \cdot a^2 & a \leq \delta \\ \delta \cdot a - 0.5 \cdot \delta^2 & \text{otherwise} \end{cases}$$

Where,

$$a = |s_{\text{pred}} - s|$$

$$\mathcal{J}_{\text{aggr}} = -\log(\sum_{i=1} p_a(op_i))$$

Final loss is the linear combination of above two losses.

#### 3.6.5 Datasets

The statistics of datasets used for training and finetuning is shown in the table 3.6.5 (Herzig et al., 2020).

We can see from table-7 (Herzig et al., 2020) that the authors used three different datasets, namely, WIKISQL (Hwang et al., 2019), WIKITQ (Pasupat and Liang, 2015), and SQA (Iyyer et al., 2017). All three are different in some aspects, and all three are the same in some aspects. For example, WIKISQL has a logical form, whereas WIKITQ and SQA do not have a logical form. SQA is conversational, while WIKISQL and WIKITQ are not conversational. WIKISQL and WIKITQ contain aggregation operators, while SQA does not. We can see authors used datasets that vary in nature.

#### 3.6.6 Results

Table-6 (Herzig et al., 2020) shows the accuracy of the WIKITQ dataset. We can see that TAPAS (pre-trained on SQA) gives state-of-the-art accuracy.

### 3.7 Knowledge Graph Embedding Based Question Answering

Knowledge Graph Embedding Based Question Answering (KGEQA) (Huang et al., 2019) aims to solve the question-answering task on the knowledge graph. The approach we will discuss here aims to answer simple questions on the knowledge graph. First, we should understand what the definition of the simple question is.

The question can be answered unambiguously if we know the head entity and predicate. The answer is the tail entity. Suppose we ask "When was Dhoni born?" from the knowledge graph. This can be answered if we know the head entity ("Dhoni") and the predicate ("born"), then the answer is the tail entity ("7 July 1981"). This is an example of a simple question. We will discuss an algorithm (Huang et al., 2019) to answer the simple question, which uses the knowledge graph embeddings.

Datasets Before understanding the algorithm, first, we should understand the data which we need for training this algorithm. We need a knowledge graph (from which the questions will be answered), we need the simple questions (input to the model for training), and we also need the triplet (head entity, predicate/relation, tail entity) for training the model.

| | FB2M | FB5M | SimpleQuestions |
|---|---|---|---|
| # Training | 14,174,246 | 17,872,174 | 75,910 |
| # Validation | N.A. | N.A. | 10,845 |
| # Test | N.A. | N.A. | 21,687 |
| # Predicates ($M$) | 6,701 | 7,523 | 1,837 |
| # Entities ($N$) | 1,963,130 | 3,988,105 | 131,681 |
| Vocabulary Size | 733,278 | 1,213,205 | 61,336 |

Figure 13: Statistics of the Dataset

Figure 13 (Huang et al., 2019) show the statistics of the dataset. Freebase is used for the knowledge graph, the simple question which can be answered if one knows the head entity and the predicate of the free base is also given with the dataset.

Our goal is to find the head entity and the predicate from the knowledge graph. Then, we can easily find the tail entity and thus our answer.

**Architecture**

The basic idea of the algorithm is simple. We need the head entity and predicate from the question. The first knowledge graph is embedded into a vector of lower dimensions. From the question, we predict the head entity embedding and the predicate embedding. We can calculate/predict the tail entity embedding using the entity embedding and the predicate embedding (link prediction task). Now the triplet/fact $(h, r, t)$ closest to this predicted triplet is chosen as the answer. Figure 14 (Huang et al., 2019) shows the architecture of the model. We can easily see the above steps in graphical format. Predicate learning, Head entity learning, Head entity detection model, and finding the closest facts are the most important component of the model. In the next sections, we will look into that.

#### 3.7.1 Predicate and Head Entity Learning Model

Basically, we have to give questions as input to the model. The supervision of the model is the predicate embedding (in the case of the Predicate learning model) and the Head entity (in the case of the Head entity learning model).

We can use any neural network model. Authors used Bi-directional LSTM with attention to predict the required embedding.

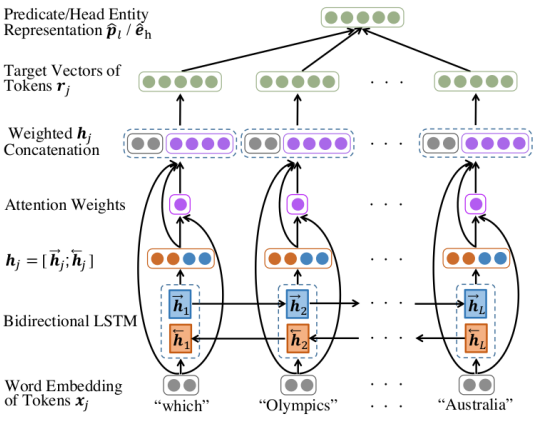


Figure 15: Architecture of the Predicate/Head Entity learning model

Figure 15 (Huang et al., 2019) shows the model's architecture. the layers of the forward-LSTM and backward-LSTM are concatenated, attention weights are calculated, then the con-

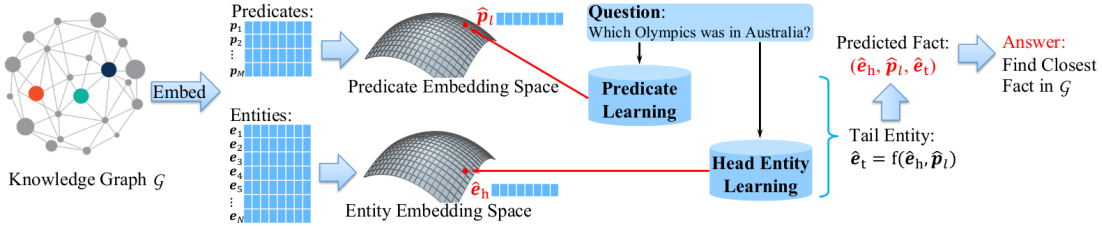


Figure 14: Overall Architecture

text vector is computed. It is concatenated with the original word embedding (residual connection). The average is taken element-wise to get the predicted embeddings.

$$\alpha_j = \frac{\exp(q_j)}{\sum_{i=1}^{L} \exp(q_i)},$$
$$q_j = \tanh(\mathbf{w}^\top [\mathbf{x}_j; \mathbf{h}_j] + b_q)$$

$\alpha_j$ are the attention weights.

$$\hat{\mathbf{p}}_\ell = \frac{1}{L} \sum_{j=1}^{L} \mathbf{r}_j^\top$$

### 3.7.2 Head Entity Detection Model

There are 15 Million entities in the Freebase knowledge graph. This may lead to a problem. Remember that the last step of the algorithm is to find the fact closest to the predicted fact. Search space will be huge in the large knowledge graphs.

The authors use a reasonable assumption (Huang et al., 2019) to solve this problem. The Head entity will be from the question itself. This assumption is valid, but not every word in the question can be a head entity. The Head Entity Detection model detects which word in the question can be the head entity.

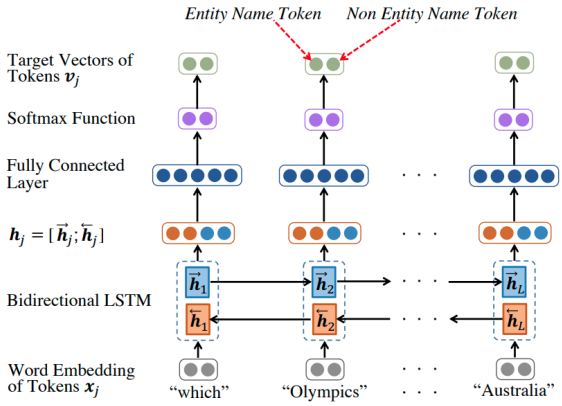


Figure 17: Head Detection Model

Figure 17 (Huang et al., 2019) shows the architecture of the head entity detection model. This is the typical bi-directional LSTM architecture with no attention. The last layer consists of two neurons (Indicating whether the token can be an Entity name token or not)

### 3.7.3 Joint Distance Metric

Now we have the predicted head entity, predicate entity, and set of all possible head entities. We must find the fact in the knowledge graph closest to the expected point. We need an objective function to see that. The first set of candidate facts is collected based on the output of the head entity detection model. Let us call the set of candidate facts by $C$. The objective function used by the author (Huang et al., 2019) is :

$$\underset{(h,\ell,t)\in C}{\text{minimize}} \quad \|\mathbf{p}_\ell - \hat{\mathbf{p}}_\ell\|_2 + \beta_1 \|\mathbf{e}_h - \hat{\mathbf{e}}_h\|_2 + \beta_2 \|f(\mathbf{e}_h, \mathbf{p}_\ell) - \hat{\mathbf{e}}_t\|_2 - \beta_3 sim[n(h), \text{HED}_{\text{entity}}] - \beta_4 sim[n(\ell), \text{HED}_{\text{non}}],$$

The first three terms find the fact closest to the predicted fact. The last two terms compute the similarity between the two words (one predicted by the head entity model and the

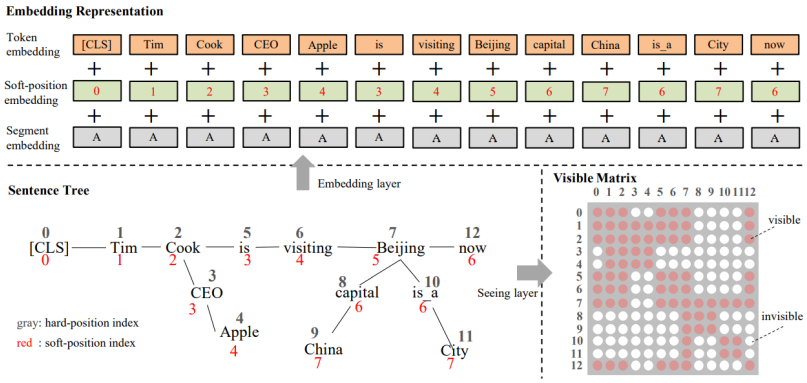


Figure 16: Embedding Layer

other the candidate fact).

The whole algorithm is described in the figure 18 (Huang et al., 2019).

**Algorithm 1:** The proposed KEQA framework

**Input:** $\mathcal{G}$, predicates' and entities' names, $\mathbf{P}$, $\mathbf{E}$, $Q$, a new simple question $Q$.
**Output:** head entity $h^*$ and predicate $\ell^*$.
/* Training the predicate learning model: */
**for** $Q_i$ *in* $Q$ **do**
  Take the $L$ tokens of $Q_i$ as the input and its predicate $\ell$ as the label to train, as shown in Figure 2;
  Update weight matrices $\{\mathbf{W}\}$, $\mathbf{w}$, $\{\mathbf{b}\}$, and $b_q$ to minimize the objective function $\|\mathbf{p}_\ell - \frac{1}{L}\sum_{j=1}^{L}\mathbf{r}_j^\top\|_2$;
/* Training the head entity learning model: */
**for** $Q_i$ *in* $Q$ **do**
  Take the $L$ tokens of $Q_i$ as the input and its head entity $h$ as the label to train, as shown in Figure 2;
  Update weight matrices and bias terms to minimize the objective function $\|\mathbf{e}_h - \frac{1}{L}\sum_{j=1}^{L}\mathbf{r}_j^\top\|_2$;
/* Training the HED model: */
**for** $Q_i$ *in* $Q$ **do**
  Take the $L$ tokens of $Q_i$ as the input and its head entity name positions as the label to train;
  Update weight matrices and bias as shown in Figure 3;
/* Question answering processes: */
Input $Q$ into the predicate learning model to learn $\hat{\mathbf{p}}_\ell$;
Input $Q$ into the head entity learning model to learn $\hat{\mathbf{e}}_h$;
Input $Q$ into the HED model to learn $\text{HED}_{\text{entity}}$ and $\text{HED}_{\text{non}}$;
Find the candidate fact set $C$ from $\mathcal{G}$, based on $\text{HED}_{\text{entity}}$;
For all facts in $C$, calculate the fact $(h^*, \ell^*, t^*)$ that minimizes the objective function in Eq. (9).

Figure 18: Algorithm KGEQA

## 3.8 K-BERT

Integrating knowledge graphs and deep learning system is vital for many NLP tasks such as question-answering, sentiment analysis, etc. This section will discuss one method to incorporate the knowledge graph with a deep learning system.

### 3.8.1 Methodology

Integrating knowledge graphs and deep learning system is a challenging task. To incorporate both systems, K-BERT uses four different modules. The first module is the knowledge layer, embedding layer, seeing layer, and mask transformers.

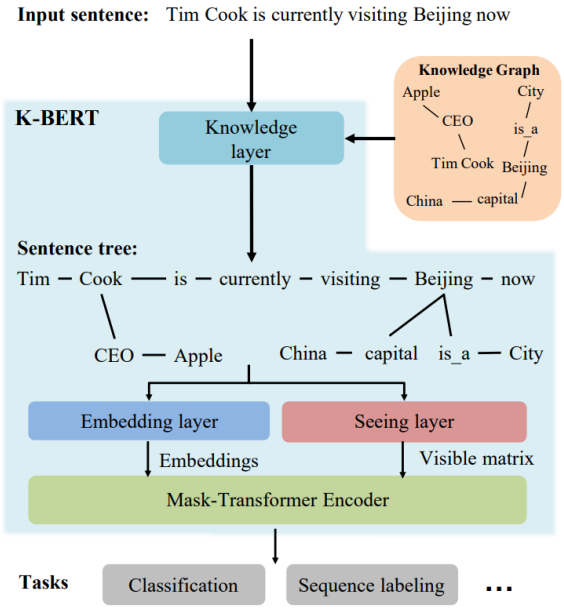


Figure 20: Architecture of KBERT

Figure 20 (Liu et al., 2020) shows the architecture of K-BERT. We can see that it uses four different modules. The knowledge layer embeds knowledge into a text using any knowledge graph. We can see an example in figure 20, if the input sentence is "Tim Cook is currently visiting Beijing now," then the sentence tree is formed using the knowledge graph.

| Models\Datasets | Book_review | | Chnsenticorp | | Shopping | | Weibo | | XNLI | | LCQMC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Dev* | *Test* | *Dev* | *Test* | *Dev* | *Test* | *Dev* | *Test* | *Dev* | *Test* | *Dev* | *Test* |
| Pre-trainied on WikiZh by Google. | | | | | | | | | | | | |
| Google BERT | 88.3 | **87.5** | 93.3 | 94.3 | 96.7 | 96.3 | 98.2 | 98.3 | 76.0 | 75.4 | 88.4 | 86.2 |
| K-BERT (HowNet) | 88.6 | 87.2 | **94.6** | **95.6** | **97.1** | **97.0** | 98.3 | 98.3 | **76.8** | **76.1** | **88.9** | 86.9 |
| K-BERT (CN-DBpedia) | **88.6** | 87.3 | 93.9 | 95.3 | 96.6 | 96.5 | 98.3 | 98.3 | 76.5 | 76.0 | 88.6 | **87.0** |
| Pre-trained on WikiZh and WebtextZh by us. | | | | | | | | | | | | |
| Our BERT | **88.6** | 87.9 | 94.8 | 95.7 | 96.9 | **97.1** | 98.2 | 98.2 | 77.0 | 76.3 | 89.0 | 86.7 |
| K-BERT (HowNet) | 88.5 | 87.4 | **95.4** | 95.6 | 96.9 | 96.9 | 98.3 | **98.4** | **77.2** | **77.0** | **89.2** | **87.1** |
| K-BERT (CN-DBpedia) | 88.8 | 87.9 | 95.0 | **95.8** | **97.1** | 97.0 | 98.3 | 98.3 | 76.2 | 75.9 | 89.0 | 86.9 |

Figure 19: K-BERT Results

Branches of that tree contain nodes and relations from the knowledge graph.

Figure 16 (Liu et al., 2020) shows the embedding layer, instead of hard positioning, K-BERT uses soft positioning. Hard positioning is giving numbers to words in the input sentence, which BERT does. Soft positioning gives numbers to branches also. It considers every branch a new sentence. Visible matrix is constructed using the soft positioning as shown in the figure 16. Third layer which is seeing layer also uses the soft positioning, as the name suggests which word can "see" other word is determined by the seeing layer. A matrix is created based on whether the word is in same branch or not.

$$M_{ij} = \begin{cases} 0 & w_i \ominus w_j \\ -\infty & w_i \oslash w_j \end{cases}$$

Where, $w_i \ominus w_j$ denotes $w_i$ and $w_j$ are on the same branch. $w_i \oslash w_j$ denotes they are in different branch. Last layer is mask-self-attention, which is an extension of self-attention. Formally, the mask-self-attention is

$$\begin{aligned} Q^{i+1}, K^{i+1}, V^{i+1} &= h^i W_q, h^i W_k, h^i W_v \\ S^{i+1} &= \text{softmax}\left(\frac{Q^{i+1} K^{i+1\top} + M}{\sqrt{d_k}}\right), \\ h^{i+1} &= S^{i+1} V^{i+1} \end{aligned} \quad (1)$$

where $W_q$, $W_k$ and $W_v$ are trainable model parameters. $h_i$ is the hidden state of the i-th mask-self-attention blocks. $d_k$ is the scaling factor. $M$ is the visible matrix calculated by the seeing layer. Intuitively, if $w_k$ is invisible to $w_j$ , the $M_{jk}$ will mask the attention score $S_{i+1}$ to 0, which means $w_k$ make no contribution to the hidden state of $w_j$.

### 3.8.2 Experiments

Pretraining is done using various datasets like WikiZh, and WebtextZh. the authors used three different knowledge graph for experiments namely, CN-DBpedia, HowNet, and MedicalKG. Various open domain and domain specific tasks are used to evaluate the K-BERT.

Figure-19 shows the result of the K-BERT. We can see K-BERT outperforms BERT in almost all the tasks except BookReview task, which is a open-domain task. K-BERT is specially designed for domain specific tasks. Therefore from the results we can conclude that integration of knowledge graph and deep learning improves the performance of the whole system.

## 4 Summary

Information Retrieval and Question Answering are important downstream tasks in NLP. This paper shows some of the work done in these fields. First, we saw sentence transformers trained using the siamese network. We saw that sentence transformer reduces the time complexity of inference. We also covered three objective functions (Classification Objective Function, Regression Objective Function, and Triplet Objective Function). Next, we described Condenser architecture, which is used to pretrain the model such that the CLS token gives powerful sentence embeddings. We also saw the coCondenser model, which uses contrastive loss with MLM objective. The embeddings generated using this technique give even more powerful sentence embeddings. Next, we described UnifiedQA, which is a single model for handling all types of formats of ques-

tions. Twenty datasets are used for training the model. T5 model is trained using these 20 datasets, and unifiedQA produces better results than all the dedicated models combined.

## References


Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Luyu Gao and Jamie Callan. 2021. Unsupervised corpus aware language model pre-training for dense passage retrieval. *arXiv preprint arXiv:2108.05540*.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 105–113.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2020. K-bert: Enabling language representation with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2901–2908.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

A Chandra Obula Reddy and K Madhavi. 2017. A survey on types of question answering system. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 19(6):19–23.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Tahseen Sultana and Srinivasu Badugu. 2020. A review on different question answering system approaches. *Advances in decision sciences, image processing, security and computer vision*, pages 579–586.