# Hierarchical Contextualized Representation Models for Answer Type Prediction

Natthawut Kertkeidkachorn[⋆1], Rungsiman Nararatwong[⋆2], Phuc Nguyen[2], Ikuya Yamada[3], Hideaki Takeda[2], and Ryutaro Ichise[2,1]

[1] National Institute of Advanced Industrial Science and Technology, Tokyo 135-0064, Japan
[2] National Institute of Informatics, Tokyo 101-8430, Japan
[3] Studio Ousia, Tokyo 100-0004, Japan
n.kertkeidkachorn@aist.go.jp, ikuya@ousia.jp
{rungsiman, phucnt, takeda, ichise}@nii.ac.jp

**Abstract.** SeMantic AnsweR Type prediction (SMART) challenge proposed a task to determine the types of answers given natural language questions. Understanding answer types play a crucial role in question answering. In this paper, we present Hierarchical Contextualized-based models, namely HiCoRe, for the SAMRT task. HiCoRe builds on top of state of the art contextualized-based models and the hierarchical strategy to deal with the hierarchical answer types. The SMART results show that HiCoRe obtains promising performance for answer type prediction on DBpedia and Wikidata datasets.



## 1 Introduction

Question Answering is one of the challenging problems in the Natural Language Processing field. The goal of question answering is to answer a question given in natural language text. Nevertheless, a natural language question is ambiguous and can lead to more than one plausible interpretation. Lack of understanding of the expected interpretation of the question results in misunderstanding the question. Question or answer type classification plays an important role in avoiding misinterpretation in question understanding. Generally, question type can be classified into a Wh-words question (Who, What, When, Where, Which, Whom, Whose, Why), while the answer type is determined by the expected type of the answer based on the question. Classifying the answer type is much more challenging than the question type due to the variety in answer type.

The SeMantic AnsweR Type prediction (SMART) challenge [5] benchmarks and investigates the answer type annotation problem. In the SMART challenge, given natural language questions, the task is to classify the granular types of

⋆ Rungsiman Nararatwong and Natthawut Kertkeidkachorn contributed equally to this project.

**Table 1.** Example questions and expected answer types from DBpedia ontology and Wikidata from the SMART dataset

| Question | Answer Type | |
|---|---|---|
| | DBpedia | Wikidata |
| Give me all actors who were born in Berlin. | dbo:Actor | Q33999 |
| Who is the heaviest player of the Chicago Bulls ? | dbo:Athlete | Q3665646 |
| What is the DMOZ ID for Glasgow ? | string | string |
| How many organizations work for Environmentalism ? | number | number |
| When did Claude Monet move to Giverny ? | date | date |
| Is the apparent magnitude of the Messier 2 less than 5.2 ? | boolean | boolean |

answers. The answer type contains 3 main category types: Boolean, Literal, and Resource. Boolean does not contain any subtypes, while Literal and Resource can be classified into fine-grained types. For Literal, there are 3 fine-grained: Number, Date, and String. For Resource, fine-grained types have corresponded to the target ontology. In the SMART dataset, DBpedia [2] and Wikidata [6] are selected as the the target ontology. In DBpedia, there are 760 coarse-grained types, while Wikidata contains more than 50,000 coarse-grained types. Table 1. illustrates example questions and expected answer types from DBpedia ontology and Wikidata ontology. The answer types can be multiple types. For example, given the question "Who is the heaviest player of the Chicago Bulls?", the expected answer types are listed as the following list: [dbo:BasketballPlayer, dbo:Athlete, dbo:Person and dbo:Agent.][4]

In this paper, we propose the Hierarchical Contextualized Representation Models, namely HiCoRe, for the answer type prediction. Our approach utilizes advanced contextualized word representation models together with the hierarchical strategy to deal with the hierarchical type of the ontology in the SMART task.

The rest of the paper is organized as follows. We describe our approach in Section 2. In Section 3, the experimental setup and the experimental results are reported. Related works are discussed in Section 4. In Section 5, we conclude our work.

## 2 Approach

The hierarchical structure of ontology requires a classification method that recognizes multi-layer labeling, including relations among the labels. Therefore, we created a stack of groups of classifiers for all levels (depth) of the ontology. Suppose an ontology $O$ consists of classes $c_{i,j} \in C$, where $i$ indicates the level where classes $c_{i,j}$ belong, and $j$ denotes each class on the $i^{th}$ level. A classifier $m_{i,k_n} \in M, 1 \leq k_n \leq z_n$ is responsible for predicting a subset of classes $c_i$.

[4] dbo: is http://dbpedia.org/ontology/

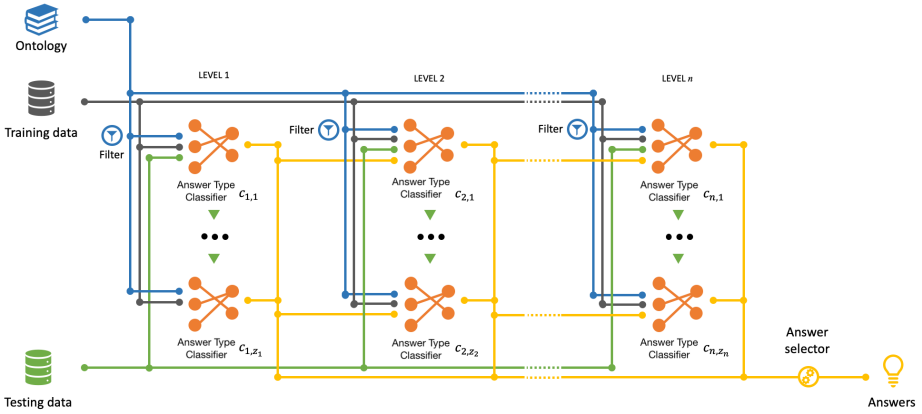


**Fig. 1.** Overall design of our architecture.

There may be single or multiple classifiers at each level, depending on configuration. The classifiers can also be of the same or different types; they operate independently and are individually customizable.

The overall architecture, as shown in Figure 1, is a modular pipeline where the ontology and training data flow through the process to train all of the classifiers. The intuition is for every level to have some very accurate classifiers that are responsible for a few classes with a large amount of training data, as well as some less accurate classifiers with more classes to classify or fewer data to train. Since we will always know the distribution of the training data with regard to classes prior to training, we created a filtering function that assigns classes to the classifiers based on pre-defined thresholds. For example, our thresholds for the first level of the DBpedia dataset are 400, 100, and 50. With this setting, our classifier $m_{1,1}$ would classify classes *dbo:Place*, *dbo:Agent*, *dbo:Work* since each appears in the training data at least 400 times.

The filtering function uses the ontology to select relevant questions, i.e., those with at least one answer (class) that the target classifier can classify. Since part of the data may not satisfy any of the filtering function's conditions, we may unintentionally ignore a portion of the training data. Thus, there should to be a default classifier that processes the rest of the data if possible, either at every level or independently. While the training data may overlap among the classifiers on the same level – resulting in increased training time – this method ensures that we feed all relevant data to every classifier, thus maximizing the accuracy.

The testing data flow through the pipeline to all classifiers differently than the training data. During testing, since we can only learn the likely answers from predictions, we may need the classifiers to perform their tasks sequentially from the first level to the last for selective testing. This method can speed up the testing process if we expected the classifiers at lower levels to be less accurate due to less training data or more classes to classify, and, therefore, should rely on the outcomes from a higher level to make predictions. Alternatively, every

classifier may make predictions on all questions; in which case, at the end of the entire process, the answer selector chooses final answers based on predefined policy.

At each level, in cases where some questions have multiple same-level answers that require a combination of classifiers to predict, the same-level classifications should not be sequential unless constrained by computing resources or any other limitations. On the other hand, sequentially performing classifications may yield better results if there are no answers that belong to the same level – for the entire dataset or parts of it – and the classifiers at higher ranks $(0, 1, ...)$ are better at predicting than the lower ranks $(..., z-1, z)$. All in all, it is up to human judgment and experimentation to decide what classifiers to use and how they should interact with each other.

### 2.1 Answer Type Classifier

**Multi-class Classification** We fine-tuned Bidirectional Encoder Representations from Transformers (BERT) [4] for our classification tasks. BERT performs outstandingly well as a base model for transfer learning across various NLP tasks. For sequence classification such as ours, we paid our attention solely to each sequence's aggregate representation, which corresponds to the first token ([CLS]) of the sequence. In other words, we used BERT to create a vector representation of each question, then turned it into an input for our down-stream classification task.

Following the instruction described in BERT's original paper, we used BERT's final hidden vector $C \in R^H$ as a sequence representation. The multi-class classifier consists of a single classification layer with weights $W \in R^{K \times H}$, where K is the number of labels. We computed the classification loss as $log(softmax(CW^T))$. The loss function restricts the use of a multi-class classifier in our pipeline to classifications that only expect a single answer, meaning that it will not be suitable for any parts of the pipeline where there can be multiple answers. On the other hand, any groups of consequent same-level classifiers, where each classifier expects a single answer, may take advantage of the sequential classification we mentioned earlier to improve the overall accuracy.

**Multi-label Classification** Our multi-label classifier is also a fine-tuned BERT model similar to the multi-class classifier. The only difference is its loss function, which we use $\sigma(CW^T)$ instead of SoftMax to allow the classifier to output multiple answers. Unlike multi-class classification, multi-label classification should not be part of selective testing, i.e., sequential classification.

### 2.2 Answer Selector

For the DBpedia dataset, we used DBpedia Lookup service [5] to find DBpedia URIs of relevant keywords. We used Natural Language Toolkit (NLTK) plat-

[5] https://wiki.dbpedia.org/lookup

form[6] for Python to extract nouns and adjectives as the keywords and retrieved the URIs for post-processing. DBpedia Lookup provides the URIs of not only keywords in a query but other similar ones as well. Using the outputs without any filtering will likely mix irrelevant answers into the correct ones. Therefore, we built a filtering function that adds a set of answers for every returned keyword from the service only if at least one of the answers match what the models in the pipeline have predicted.

Another post-processing task for both datasets is answer selection. We defined three selection strategies, which are *top-down bottom-up* and *independent*. The *top-down* strategy prioritizes answers at higher levels. It includes lower-level answers only if their parents are present. The *bottom-up* strategy does the opposite; it traces the branch where the answer belongs to the top level and adds all elements on that branch as the answers. The *independent* strategy does not change the answers.

## 3 Experiments and Results

In this section, the experimental setup and results are presented. The details of the expreiments are as follows.

### 3.1 Experimental Setup

In the experimental setup, we present Dataset, Experiment Setting and Evaluation Metrics.

**Datasets.** The SMART task consists of two datasets: DBpedia and Wikidata. In DBpedia dataset, the target ontology is DBpedia ontology, while in Wikidata the target ontology is Wikidata. The statistical details of the SMART dataset are listed in Table 2. Since both datasets do not provided the validation set, we randomly selected 10% of the training set in both datasets to construct the validation set.

**Settings.** We experiment on many contextualized-based models, including distilbert-base-uncased, bert-base-uncased, bert-large-uncased, roberta-base, and roberta-large to train the answer type classifier. We implement the contextualized-based models by using the hugging face repository [7]. Then, we manually set hyper-parameters then test on the validation set to find a reasonable set of hyper-parameters. As a result, we set the hyperparameters as follows: batch: 16, learning rate: $5e-5$, epochs: 10-45, dropout rate: 0.1.

Before training, we studied the distributions of training data with regard to classes (labels) at each level and found a similar pattern across all levels in both datasets. As shown in Figure 2, there are generally a few classes with a large amount of training data, while the rest only have a few to train. Therefore, for every level, we created a set of classifiers based on how much information we

[6] https://www.nltk.org
[7] https://huggingface.co/models

**Table 2.** The Statistics of the SMART dataset on DBpedia and Wikidata

| Train set | | |
|---|---|---|
| | DBpedia | Wikidata |
| Boolean | 2,799 | 2,139 |
| Literal | 5,188 | 4,429 |
| - Nunber | 1,634 | 0 |
| - Date | 1,486 | 0 |
| - String | 2,086 | 0 |
| Resouce | 9,584 | 11,683 |
| Total | 17,571 | 18,251 |
| Test set | | |
| | DBpedia | Wikidata |
| Total | 4,369 | 4,571 |

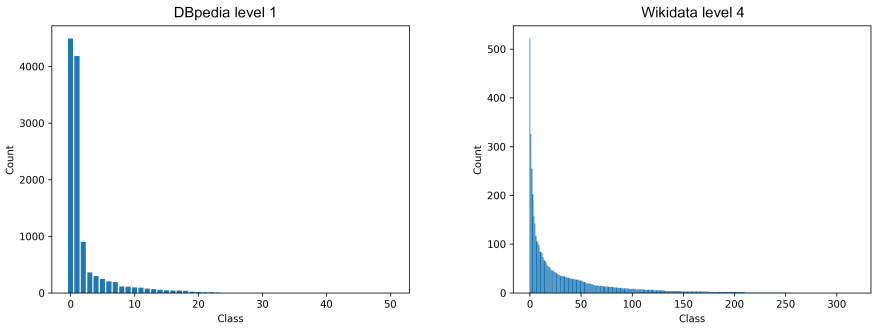


**Fig. 2.** Distribution of the training data at level 1 (DBpedia) and level 4 (Wikidata).

have to train them. For DBpedia, we created up to three classifiers per level with thresholds of 400, 100, and 50, meaning that any classes with at least 400 training samples will be included in the first classifier and so on. The thresholds for Wikidata are 1000, 300, 100, and 50.

**Evaluation Metrics.** In the fine-tuning process on the validation set, we use standard Accuracy, F1-macro, F1-weighted by the sklearn library [8] for the category classification, while only F1-macro and F1-weighted are used to evaluate the resource types on each level of the hierarchy in the ontology. We use these metrics to find the hyperparameters that are the best suit for each level. Due to the structure of the ontology in the datasets, there are five levels in DBpedia and 11 levels in Wikidata.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report

**Table 3.** Results of the SMART Task on the validation set (10 % of the training data)

| Level | DBpedia | | | Wikidata | | |
|---|---|---|---|---|---|---|
| | Accuracy | F1-macro | F1-weighted | Accuracy | F1-macro | F1-weighted |
| Category | 0.9876 | 0.9681 | 0.9875 | 0.9858 | 0.9606 | 0.9858 |
| Level-1 | - | 0.9287 | 0.9787 | - | 0.8937 | 0.9511 |
| Level-2 | - | 0.9467 | 0.9839 | - | 0.8845 | 0.9626 |
| Level-3 | - | 0.9223 | 0.9751 | - | 0.9011 | 0.9640 |
| Level-4 | - | 0.9416 | 0.9729 | - | 0.8526 | 0.9457 |
| Level-5 | - | 1.0000 | 1.0000 | - | 0.9020 | 0.9530 |
| Level-6 | - | - | - | - | 0.8588 | 0.9644 |
| Level-7 | - | - | - | - | 0.9163 | 0.9614 |
| Level-8 | - | - | - | - | 0.8939 | 0.9402 |
| Level-9 | - | - | - | - | 0.9364 | 0.9662 |
| Level-10 | - | - | - | - | 0.9380 | 0.9564 |

**Table 4.** Results of the SMART Task on the test set

| DBpedia | | | Wikidata | |
|---|---|---|---|---|
| Accuracy (Category) | nDCG@5 | nDCG@10 | Accuracy (Category) | MRR |
| 0.964 | 0.749 | 0.721 | 0.96 | 0.59 |

For the final evaluation on the test set, we follow the metrics provided by the SMART challenge [9]. In the SMART challenge, the evaluation metrics are varied due to the dataset. In DBpedia, the category accuracy and normalized discounted cumulative gain (nDCG) are used. The nDCG is set at 5 (nDCG@5) and 10 (nDCG@10). The evaluation metrics in Wikidata are the category accuracy and mean reciprocal rank (MRR).

### 3.2 Results

Table 3 listed the results of HiCoRe on the validation set of each level with the best setting we found, while Table 4 reported the results on the test set of the SMART task. The results in both tables show that we could achieve high accuracy (more than 96%) for the category prediction. Furthermore, on the validation set HiCoRe gives promising F1-macro and F1-weighted scores (more than 80%) in both DBpedia and Wikidata.

## 4 Related Works

Answer type classification could be viewed as the entity type classification, where the answer to the query question is given as the entity. There are many research

[9] https://smart-task.github.io

works [1, 7, 8] related to an entity type in the NLP community. Nevertheless, the SMART dataset does not provide the answers to the query questions. Therefore, predicting the answer type is much more challenging than the conventional entity type classification due to the absence of the answer entity. There is a study investigating answer type prediction with the same setting as the SMART dataset. In the study [3], the type matcher is applied on the question to get attention words for building the classifier based on the syntactic structure features. Nonetheless, this work does not consider the hierarchical structure of answer types.

## 5 Conclusion

In this paper, we introduce a novel method using hierarchical contextualized representation models, named HiCoRe, for answer type prediction. HiCoRe adopts state of the art contextualized word representations together with the hierarchical strategy to deal with the answer type prediction. In HiCoRe, we investigate varieties of BERT classifiers, which could be configured on each hierarchical level. By fine-tuning BERT-based models in HiCoRe, we could reach promising results on the SMART dataset. Future improvement may include data augmentation and question-answer generation for training, especially for classes with fewer examples. The source code is available at https://github.com/rungsiman/smart.

## References

1. Abhishek, A., Anand, A., Awekar, A.: Fine-grained entity type classification by jointly learning representations and label embeddings. pp. 797–807. Association for Computational Linguistics, Valencia, Spain (Apr 2017)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735. Springer (2007)
3. Bogatyy, I.: Predicting answer types for question-answering. https://cs224d.stanford.edu/reports/Bogatyy.pdf, accessed: 2020-09-25
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of NAACL-HLT. pp. 4171–4186 (2019)
5. Mihindukulasooriya, N., Dubey, M., Gliozzo, A., Lehmann, J., Ngomo, A.C.N., Usbeck, R.: SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge. CoRR/arXiv **abs/2012.00555** (2020), https://arxiv.org/abs/2012.00555
6. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)
7. Xu, P., Barbosa, D.: Neural fine-grained entity type classification with hierarchy-aware loss. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). pp. 16–25. Association for Computational Linguistics, New Orleans, Louisiana (Jun 2018)
8. Yogatama, D., Gillick, D., Lazic, N.: Embedding methods for fine grained entity type classification. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). pp. 291–296 (2015)