Better Path: Employing New Features and Encoders for Path Naturalness Prediction

A0194568L, A0194567M, A0211301J, A0149790R
Group 14
Mentored by Muhammad Reza Qorib
{e0376998,e0376997,e0493632,e0014694}@u.nus.edu

Abstract

Knowledge graphs are valuable sources of information for various artificial intelligence agents as they can efficiently provide semantic contexts based on real-world knowledge by humans. However, one challenge when using knowledge graphs is the intricacy of denoising paths from the graph — selecting one path over another is non-trivial. This work looks at training a better classification model for predicting the quality of paths between concepts and performing pairwise comparisons to choose paths more optimally. In particular, we reconsider several assumptions made in existing models during path encoding, feature engineering, and model selection. We analyze improvements to existing model architecture and new features for embedding edges. Most notably, we show that our methods of path encoding can better retain salient signals for model training. In addition, we show that incorporating path heuristics also helps in optimal path prediction task.

1 Introduction

Knowledge graphs are semantic networks of real-world information in which each vertex within a graph represents some concept, and each directed edge (v_1, v_2) between a pair of vertices v_1 and v_2 has a label specifying the relationship from vertex v_1 to vertex v_2.

Typically built using information extracted from various data sources, knowledge graphs have applications across different domains, especially in many natural language processing tasks. For example, some question-answering models (Ma et al., 2019; Yasunaga et al., 2021) and recommender systems (Wang et al., 2019; Zhou et al., 2020) use knowledge graphs to find semantic relations between concepts.

Our work is motivated by the observation that many of such models using knowledge graphs require the selection of the most optimal path between two vertices, but the performance of this task is often unsatisfactory due to incomplete and noisy knowledge graphs. For instance, publicly available knowledge graph ConceptNet (Speer et al., 2016) contains the knowledge ⟨"an apple" $\xrightarrow{\text{UsedFor}}$ "getting good grades from a teacher"⟩ in which the relation may not be clear. For many models relying on short path lengths as the main heuristic for path selection, model performance may be adversely impacted by such noises in knowledge graphs.

In this work, we consider training a model to predict the score of a path from knowledge graph based on the path quality. Using the path score, pairwise path comparison can be performed for the purpose of an improved path selection strategy given two different paths from a knowledge graph. This is motivated by the work from Zhou et al. (2019), hereafter abbreviated as ZSS, in which they managed to build a model which substantially improves the ability to select better paths than using simple heuristics, although test accuracies are still non-ideal with accuracies below 70.0%. Like ZSS, we adopt the same vague definition of path quality to be how natural a path is, as this allows flexibility in making use of a dataset well-annotated by humans which we describe in later section.

We believe this project to improve model performance is essential for optimizing a myriad of downstream tasks that utilize knowledge graphs.

We use a crowdsourced dataset from ZSS, which consists of annotated paths from ConceptNet, to train a better classification model for predicting the quality of paths between concepts. Our key contributions include reconsidering several assumptions made in existing work by focusing on natural language characteristics, such as proposing alternative methods to encode paths for retaining salient signals, exploring the use of new features, and incorporating proven heuristics for path

quality prediction.

2 Related Work

ConceptNet (Speer et al., 2016) is a natural language knowledge graph prepared that covers the majority of commonly used words in various languages, including English. It is widely used in natural language research for semantic analysis. It contains many words and phrases and represents a set of common word relations among them.

Many pairs of words in ConceptNet are not directly connected, but are typically connected via some paths, usually non-unique. The number of paths between any pair of words in ConceptNet grows exponentially as the admitted path length grows. Hence, path selection, the task to select the better paths or even the best path between pairs of words, is a non-trivial but very useful task for NLP research based on ConceptNet.

There have been some heuristics developed to assist path selection. A common approach is limited graph traversal, by total number of nodes explored (Boteanu and Chernova, 2015), or maximum path length (Lao and Cohen, 2010; Guu et al., 2015). Some other approaches with quantitative heuristic values include word embedding similarity (Gardner et al., 2014) and network flow calculated based on vertex degree (Lin et al., 2015).

Other studies to deal with noise that exists in knowledge graphs include Vassiliades et al. (2021) who looked at the same problem and observed that certain combinations of edges may form a longer path but is better than other shorter paths.

ZSS developed a path selection routine beyond heuristics. They first represent each path as a sequence of alternating encodings for vertices and edges, in order of occurrence in path. Then, they train an LSTM on the sequence to give an encoding for the entire path. Finally, they train a fully connected layer from the path encoding to a score for the path. Instead of assigning scores to paths manually in the training data, they tune the scores by pairwise comparisons, such that the objective of the score predictor is to give a better path in the pair higher score and vice versa. Our work is based on ZSS, and we will further elaborate flaws and propose modifications to their work in Section 3.

Path Pair	Majority Label
Car $\xrightarrow{\text{AtLocation}}$ **Land** $\xleftrightarrow{\text{RelatedTo}}$ **Country** $\xleftrightarrow{\text{RelatedTo}}$ **Town**	Favored
Point $\xleftrightarrow{\text{RelatedTo}}$ **Mountain** $\xleftrightarrow{\text{RelatedTo}}$ **Wave** $\xrightarrow{\text{IsA}}$ **Woman**	Unfavored
Tissue $\xrightarrow{\text{IsA}}$ **Paper** $\xleftrightarrow{\text{RelatedTo}}$ **Box**	Favored
City $\xrightarrow{\text{AtLocation}}$ **School** $\xleftrightarrow{\text{RelatedTo}}$ **House** $\xleftrightarrow{\text{RelatedTo}}$ **Box**	Unfavored
Job $\xleftrightarrow{\text{RelatedTo}}$ **Point** $\xleftrightarrow{\text{RelatedTo}}$ **Number** $\xrightarrow{\text{IsA}}$ **Size**	Favored
Doctor $\xleftrightarrow{\text{RelatedTo}}$ **House** $\xleftrightarrow{\text{RelatedTo}}$ **Street**	Unfavored

Table 1: Examples of annotated pairs of paths. For each pair of paths, human responses on which path is more natural were collated. Bold words represent vertices while arrows represent edges.

3 Methodology

ZSS have presented the strength of their model over other heuristics comprehensively, so we base our work mainly on this work, but with modifications that further improve the performance.

Training Data. We use a dataset[1], which consists of human-annotated paths from ConceptNet. This data was also used in the work by ZSS, and this allows us to compare the performance of our propose model against theirs. In Table 1, we present some annotated pairs of paths found in the dataset. Since the dataset consists of clear concepts and paths, further preprocessing was not required. Mainly, the necessary objective is to use NL methods to encode paths for training of a prediction model.

Since vertices and edges have latent semantic meaning, we focus on retaining salient signals. We maintain the overall architecture such that there are three major steps in the task:

1. **Vertex and edge encoding.** Convert vertices (words) and edges (word relations) on the given path to vectors.

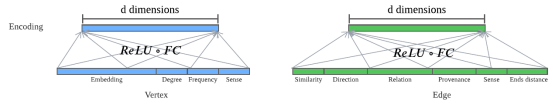
2. **Path encoding.** Use the sequence of encodings generated from the previous step to

[1]Retrieved from the GitHub repository https://github.com/YilunZhou/path-naturalness-prediction by ZSS.

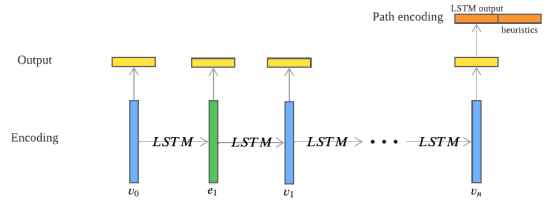
obtain an encoding for the entire path using LSTM.

3. **Score Prediction.** Use the path encoding to train a predictor of score such that the accuracy of the prediction of a better path among the path pairs for comparison is maximized.

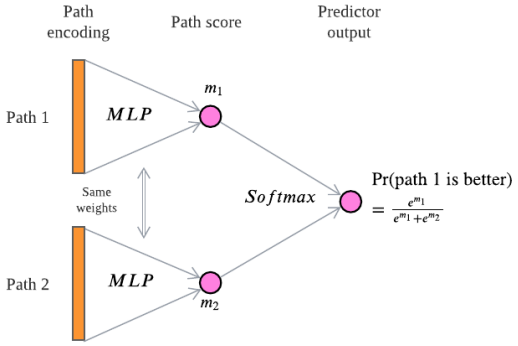
However, we make modifications to improve the performance further, with details to be detailed in the following subsections. Figure 1 shows the model architecture we adopt.



(a) Step 1. Vertex and edge encoding. Features of vertices and edges are concatenated and encoded using a ReLU-activated FC layer.



(b) Step 2. Path encoding. Vertices and edges are alternated as input to a standard LSTM. The output of LSTM is concatenated with heuristics to give the path encoding.



(c) Step 3. Path encoding is inputted to an MLP for score prediction. The score of a pair of path is then passed to a softmax layer to predict a better path within a pair. The inaccuracy for human-annotated pairwise comparison is the loss function.

Figure 1: Architecture of the model for path quality evaluation. The architecture is adapted from ZSS.

The following subsections will describe each step in detail, with our modifications and some other variants we considered introduced.

3.1 Vertex and Edge Encoding

ZSS encoded vertices and edges by first identifying a set of relative features, each with a variable dimension, and train an encoder for each feature independently using a ReLU-activated fully-connected layer so that the dimension of each feature is the same. They took the average encoding of all features as the encoding for the vertex/edge. We call this original version of encoder **BasicEncoder**.

Intuitively, each feature has different impact on signalling the goodness of a path, and hence naïvely taking the average may not be the best approach to combine features. To solve the problem, instead of training an encoding for each feature and taking the average, we first concatenate all features and train an encoding on the concatenated vector. Thus, we allow varying weights to be optimized by training on different features and even different dimensions within the same feature. We call this new encoder **ConcatEncoder**.

The encodings are to be passed to an LSTM that treats vertex and edge encodings as the same kind of data, while they have different meanings. Consequently, weights between hidden layers are shared for vertex-to-edge and edge-to-vertex, which is not an intuitive result. To get away with this problem, we propose another encoder **CombinedConcatEncoder**, which instead of treating every edge and vertex as independent entities with identical meanings (for LSTM), we concatenate a pair of adjacent vertex and edge as one entity so that for a path of length n (in terms of the number of edges), instead of constructing a sequence of alternating vertices and edges of total length $2n + 1$, we construct a sequence of edge-vertex pairs with total length $n + 1$. This encoder is visualized in Figure 2.

In the work done by ZSS, a list of experimented features useful for the task was provided, and we will adopt and attempt to replicate them in our model as well (see Appendix A). However, it is still worthwhile to explore other useful features. In particular, according to the ablation study done by ZSS, edge ends similarity is the most-contributing feature type, with improved performance significantly larger than other feature types. Intuitively, relationships between the two ends of an edge are beneficial in evaluating the goodness of paths, but edge ends similarity is the only feature that captures such a relationship. Thus, we want to extend more features for edges that evaluate the relationship between its two ends. The features we added

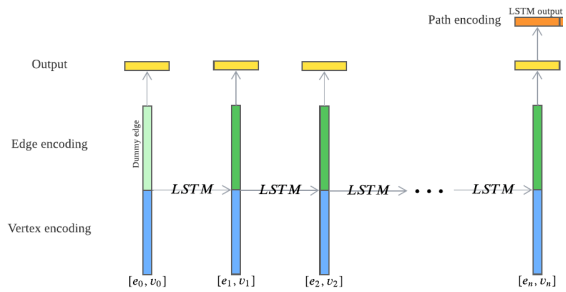


Figure 2: Architecture of **CombinedConcatEncoder**. A dummy edge is appended at the front, and then adjacent edge-vertex pairs are concatenated as one entity and passed to LSTM for training a path encoding. For a path of length n, the total number of encodings passed to LSTM is $n + 1$, which are $[e_0, v_0]$ where e_0 is dummy, $[e_1, v_1], \ldots, [e_n, v_n]$.

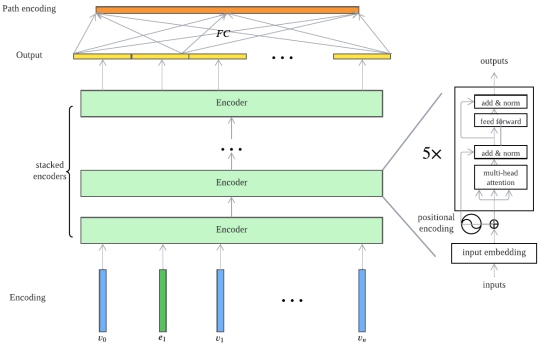


Figure 3: Architecture of Transformer Encoder. It takes in the encoding of vertices and edges. The input is passed through 6 stacked encoders. The output of the transformer is concatenated and passed through a FC layer to construct path encoding.

are L1 and L2 distances between the embeddings of the two ends of an edge.

3.2 Path Encoding

ZSS used a standard LSTM that takes in the encodings of vertices and edges to encode the entire path. Technically what is needed at this step is to encode a sequence. Hence, we will try different variants of RNN in place of LSTM as well.

With the recent dominance of Transformer in the field of NLP, adding our custom Transformer Path Encoder is another extension to the work by ZSS in an attempt to better capture vertices and edges found at the front of each path. Intuitively, using LSTM may lead to more attention on the last few vertices and edges which is not ideal for our context, since we feel that all vertices and edges are equally important. Since we want to extract out an embedding from a path instead of producing a sequential output, we only use the encoder part of the Transformer. Specifically, the Transformer Encoder we tested have several stacked Encoder, each with 5 heads. The output of the Transformer Encoder then goes through a linear layer to match the output dimension. The architecture of Transformer Encoder is seen in Figure 3.

In addition, we will augment the path encoding trained by variants of RNN with some heuristics. Although the model by ZSS outperformed heuristic approaches, the heuristics are nevertheless still proven to be useful. Therefore, we propose to combine path encoding with the use of heuristics as features for the entire paths to improve the performance of the score predictor. We concatenate calculated heuristics to the path encoding trained by variants of RNN before inputting it to the predictor. Heuristics we added include:

1. **Source-Target Similarity.** Cosine similarity of the word embeddings of the two ends of the path.

2. **Pairwise Similarity.** Average cosine similarity of every two adjacent words on the path.

3. **Resource Flow.** Sum of log of degrees of all vertices on the path. This is used as a simplified version of the network flow heuristic (Lin et al., 2015).

4. **Length.** Length of the path, defined by number of vertices in the path.

The complete path encoding is the encoding trained by variants of RNN concatenated with heuristics.

3.3 Score Prediction

ZSS trained the predictor to assign a path (encoding) an associated score. The training data contains path pairs with one of them annotated as more natural. The objective of the predictor is to give scores such that after taking softmax on the scores of the two paths in each pair, the prediction accuracy on which one is better is maximized.

ZSS used only one fully-connected layer for predicting scores out of path encoding, which assumes that the score can be calculated as a linear function of the path encoding. However, such linearity does not necessarily hold. Hence, to slightly

complicate the model to allow for complex scoring functions from path encoding, we insert another ReLU-activated fully-connected layer to the predictor.

4 Experiments

This section shows and describes experimental results obtained from our proposed modifications. The experiments we have carried out include adding new vertex/edge features, appending heuristics to the path encoding, trying out different vertex/edge encoders and path encoders, adding more layers to the predictor model and hyperparameter tuning. All experimental results can be reproduced via running functions in `code/experiment.py` under our repository[2]. Discussion of these results can be found in later section.

4.1 Dataset and Training

. We have also incorporated multiprocessing in the experiment script to speed up and save time. We use the same "science" dataset as the first setting in ZSS' work. Specifically, ZSS have previously obtained 100 words by performing a random walk on ConceptNet. Then, these 100 words were used to induce 5,864 paths with 4 or less nodes. 9833 training path pairs and 2459 testing path pairs were sampled from the 5,864 paths with naturalness annotation. During each iteration of training, a sample of size `resample_size` is extracted from the training pairs and used for training. The testing pairs are used to evaluate the performance of the fitted model at the end of each iteration. We use 4000 iterations for all our experiments.

4.2 Evaluation Metric

In the following experiments, our main evaluation metric is the accuracy of predicting which path would be favored (i.e., the path of higher quality), for a given pair of paths in the test split. This metric was chosen for the reasons below:

1. **Balanced dataset.** In each pair of paths, there is exactly one favored path and one unfavored path as annotated by humans. As such, it is reasonable to use accuracy as a metric over other metrics such as F1 score, recall, or precision.

[2]Our GitHub repository https://github.com/CS4248-Group14/Better-Path

Add both features	66.6%
Add Edge L1 Distance Feature	**66.7%**
Add Edge L2 Distance Feature	65.8%
No new features (*Baseline, ZSS*)	65.6%

Table 2: Ablation study on feature importance.

2. **Low ambiguity.** Ground truth labels are annotated by humans and quality of annotations were ensured during data collection. This results in less ambiguity than qualitative metrics.

3. **Same domain.** By testing the model on a dataset of the same domain as the training data, verification of our proposed models is consistent and this eliminates any potential noise due to non domain-specific knowledge base.

4. **Baseline comparison.** As the main purpose of this work is to reconsider modeling assumptions in the existing model from ZSS, using the same evaluation metric as the existing work allows a fair comparison.

4.3 Ablation Study on New Features

To discern the contributions of our newly added features (distance between word embeddings on the two ends of an edge), we first train our models with ZSS' original feature set and treat it as the baseline. We then insert our new distance features into the feature set, retrain the model, and compare it with the baseline.

We can see from Table 2 that both L1 and L2 distances can improve the performance. However, adding the L1 distance alone works the best, instead of adding both L1 and L2 distances.

4.4 Ablation Study on Heuristics

Similar to the feature ablation study, we also experimented the effects of heuristics. By fixing other parameters and only changing heuristics used across different runs, we obtained the results in Table 3.

We can see that all heuristics improve performance to some extent. Nevertheless, using **Pairwise Similarity** heuristic alone yields the best performance.

4.5 Experiment on Vertex/Edge Encoders

In Section 3, we proposed two modifications to the vertex/edge encoders. The first is **Conca-**

Add all 4 heuristics	66.6%
Add Source-Target Similarity Heuristic	66.1%
Add Pairwise Similarity Heuristic	**67.0%**
Add Resource Flow Heuristic	65.9%
Add Length Heuristic	65.9%
No new heuristics (*Baseline, ZSS*)	65.6%

Table 3: Ablation study on heuristic importance.

CombinedConcatEncoder	64.7%
ConcatEncoder	**66.2%**
BasicEncoder (*Baseline, ZSS*)	65.6%

Table 4: Test accuracy for different vertex/edge encoders

tEncoder, which concatenates vertex/edge features instead of averaging them. The second vertex/edge encoder is **CombinedConcatEncoder**, which pairs up each vertex with an edge and uses their combined features as the input to variants of RNN.

The effectiveness of our proposed vertex and edge encoders can be seen from the following experiment results in Table 4.

We can see that **ConcatEncoder** performs the best.

4.6 Experiment on Path Encoders

We compare the performance of Transformer Encoder with that of Vanilla RNN and LSTM. Note that modifications that are proven useful (such as adding heuristics) in the previous experiments were used in all three models (which is the reason why the LSTM's performance here is higher than the baseline LSTM with ZSS' configuration).

We can see from Table 5 that LSTM still produces the best results. However, we use six stacked Transformer Encoders in the above experiment, and we suspect it might overfit the training set. Hence we try to reduce the number of stacked encoders, and the results can be seen in Table 6.

We can observe that 3 stacked Transformer Encoders works the best, although it still falls behind LSTM. Moreover, the training time of Transformer Encoder is significantly longer than LSTM's if no GPU is used.

TransformerEncoder	65.4%
Vanilla RNN	64.4%
LSTM	**66.4%**

Table 5: Test accuracy for different path encoders

Number of stacked Transformer Encoders	
1	65.3%
3	**65.6%**
6	65.4%

Table 6: Test accuracy for different number of stacked Transformer Encoders

Multilayer	**66.6%**
Single layer (*Baseline, ZSS*)	65.6%

Table 7: Test accuracy for varying number of layers in predictor model

4.7 Multilayer

To examine the effect of multiple layers in the predictor model. We add an additional ReLU-activated linear layer to the predictor model, and compare its performance with the baseline predictor model with only one linear layer.

We can see from Table 7 that multilayer did improve the performance by 1%.

4.8 Hyperparameter Tuning

4.8.1 Word Embedding Size and Path Code Length

We performed similar grid search of word embedding dimension and path code length as ZSS' work, with ranges {2, 10, 50, 100, 300} and {2, 5, 10, 30, 50} respectively. Note that the path code length refers to the output length from the RNN variant (LSTM).

We can see from Table 8 when values for (word embedding size, path code length) pair is (300, 5) or (50, 30), the model performs the best.

4.8.2 Resample Size

Recall that for each iteration, we resample a sample of size `resample_size` from the training

		Path Code Length				
		2	5	10	30	50
Emb.Dim.	2	62.3	63.9	63.6	65.0	65.2
	10	62.2	63.7	65.1	65.7	65.3
	50	65.4	64.7	65.4	**67.0**	65.8
	100	64.8	66.4	66.3	66.5	65.9
	300	66.2	**67.0**	66.1	66.5	66.1

Table 8: Test accuracy (in percentage) while tuning hyperparameter

resample_size	
32	64.1%
64	66.0%
256	66.4%
512	**66.9%**
1024	66.2%

Table 9: Test accuracy for different resampling size

pairs and use the the sample to train the model. We test the same model with `resample_size` in {32, 64, 256, 512, 1024} to examine the effect of `resample_size`.

As observed in Table 9, the test accuracy increases as `resample_size` goes from 32 to 512. However, the accuracy decreases when `resample_size` goes from 512 to 1024.

5 Discussion

In this section, we discuss our experimental results and findings.

5.1 Importance of Heuristics and New Features

In Section 3, we discuss the rationale for augmenting the path encoding with existing heuristics. Based on the earlier discussion, four heuristics of optimal paths described by ZSS were experimented. From experimental results, any of the four heuristics can individually improve our method's performance. In particular, pairwise similarity heuristics for the entire path performs the best, improving the accuracy by 1%. This result is comparable to the ablation studies by ZSS, which found that ends similarity is the most-contributing feature type (the contribution is significantly more than other features), so we could reasonably assume that such type of two-ends information of edges is very useful.

After adding heuristics and getting the observation that two-ends information of edges is useful, we consider adding new features about two-ends edges. In the original features proposed by ZSS, the only feature about two-ends edges is the cosine similarity of the embeddings of two ends. Therefore, we add L1 distance and L2 distance features to edge's features. From experiments, both distances improve the performance. But we observe that L1 distance performs better than L2 distance. The performance increases about 1% than the baseline while L2 improves only about 0.2%.

The reason is that L2 distance calculates the square of the weights, which amplifies the outliers in the data exponentially, while L1 distance takes the absolute values of the weights, so the cost increase linearly. For word ends' embedding, the dimension is 300. So, one outlier improves the sum a lot.

Overall, adding the edge L1 distance feature and pairwise similarity heuristic both improve accuracy. The training time is almost the same. So we decide to keep L1 distance and pairwise similarity heuristic for encoding paths.

5.2 Vertex and Edge Encoder Choice

We propose two versions of new vertex and edge encoders: **ConcatEncoder** and **CombinedEncoder**. We compare the original version of encoder proposed by ZSS: **BasicEncoder** with these two new encoders. The intuition of generating these two new encoders is that we notice that the **BasicEncoder** does simply averaging the features among vertex/edge. It may not be a good assumption that every feature contributes the same to the natural path. So, we first propose **ConcatEncoder** to concatenate all features and train an encoding on the concatenated vector. However, we expected that this concatenating operation would not improve the performance much as it treats the vertex and edge feature as the same kind. Therefore, we propose **CombinedEncoder**, pairing up each vertex and its edge, whose details are in Figure 2.

However, the experiment result does not prove our assumption. **ConcatEncoder** performs the best by improving the accuracy by 0.6% while **CombinedEncoder** even perform worse than **BasicEncoder**. We believe this is attributed to the fact that pairing of vertex and its edge spoils the relative relationship between the same kinds of features, such as vertex and vertex, edge and edge. Therefore, we choose **ConcatEncoder** as our vertex and edge encoder.

5.3 Path Encoder Choice

We compare three variants of RNN in terms of their capabilities to encode paths. The Vanilla RNN with simple hidden states performs the worst. This is expected as the Vanilla RNN lacks the capability of preserving long-term dependencies. Since we are only taking the last hidden states from the Vanilla RNN (same as LSTM), the output path encoding from the Vanilla RNN might not be able to retain the information of the first

few vertices/edges. On the other hand, by using LSTM, information about the beginning part of a path is better preserved using the memory cell in each LSTM unit.

Although we had high expectations for the Transformer Encoder, it does not outperform the LSTM model. We initially suspected that the Transformer Encoder used in the initial edge encoder experiment might have too many encoder layers (6 stacked encoders) to overfit the training samples. However, after experimenting with a different number of encoder layers, we see few improvements, and the overall performance is still worse than LSTM's. This phenomenon is because our paths are generally very short (with only four vertices). Hence, Transformer's powerful self-attention mechanism, which allows the model to attend to different parts of an arbitrarily long input sequence effortlessly, cannot fully exhibit its capabilities here. LSTM's memory cell is sufficiently useful for encoding full sequences' information for short sequences like our paths.

Additionally, the training time of Transformer Encoder is significantly longer than LSTM's using CPU. This is expected because, without GPU's powerful parallel matrix computation, Transformer Encoder's main advantage of not requiring computing path encoding sequentially is not exploited. Therefore, we adapt the code to utilize GPU and run the code on an HPC. The training time is much more reasonable, although it is still longer than LSTM's. This might arise because we are using several encoders stacked on top of each other.

Overall, LSTM outperforms Transformer Encoder in our context in terms of test accuracy and training time, and we decide to keep it as our path encoder.

5.4 Multilayer Effect

We observe that ZSS employed a single layer in the predictor model. While the single layer can only learn linear functions, multiple layers can learn non-linear functions. So we choose to add an activated layer to predictor model. Among many activation functions, ReLU has advantages that it is fast to compute, and it does not suffer from vanishing, like $\tanh$ functions. For results, we expected that after adding ReLU functions, the accuracy would improve compared with the baseline in the work by ZSS.

The result conforms to our hypothesis given that the accuracy improves by 1%. Additionally, the training time does not show much difference from the baseline. This can be attributed to how ReLU can be computed more easily since its derivative is constant for bigger and smaller inputs than zero.

6 Conclusion

In this work, we propose novel methods to challenge assumptions made in the work by ZSS to train a model for predicting the score of a path from knowledge graph based on the path quality. We successfully train a better classification model by employing alternative methods to encode paths using new features and incorporating proven heuristics. The main insights we obtained are as follows:

1. Though some new features can outperform previously proposed heuristics, heuristics need to be kept for a better classification.

2. Among all features, a path's adjacency vertices' relation contributes the most.

3. Instead of using average features, concatenating features performs better.

Our work still has some limitations. For example, we only add one ReLU-activated fully-connected layer the predictor. Nevertheless, two more hidden layers rather than one hidden layer may still have the potential to give a better performance. For loss function, we choose NLL, and optimization algorithm is Adam, which is consistent with ZSS. We can fine-tune hyperparameters in Adam in future studies and try other loss functions. Also, **CombinedEncoder** does not perform as well as we thought. In the future, one possible direction is to find a method to find a relation between vertex and edge, and combine and encode vertex and edge features without losing their separate characteristics. Due to time constraints, we could not test our model's performance on downstream tasks such as question-answering, but this can be a potential future work to evaluate the improvements we show in this work.

References

Adrian Boteanu and Sonia Chernova. 2015. Solving and explaining analogy questions using semantic networks. In *AAAI Conference on Artificial Intelligence*, AAAI, pages 1460–1466.

Junpeng Chen and Juan Liu. 2011. Combining conceptnet and wordnet for word sense disambiguation. In *International Joint Conference on Natural Language Processing (IJCNLP)*, volume Proceedings of 5th International Joint Conference on Natural Language Processing, pages 686–694, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Matt Gardner, Partha Talukdar, Jayant Krishnamurthy, and Tom Mitchell. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 397–406, Doha, Qatar. Association for Computational Linguistics.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. arXiv.

Ni Lao and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53—-67.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Raw, and Song Liu. 2015. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the Conference Empirical Methods in Natural Language Processing (EMNLP)*, pages 705–714, Lisbon, Portugal. Association for Computational Linguistics.

Kaixin Ma, Jonathan Francis, Quanyang Lu, Eric Nyberg, and Alessandro Oltramari. 2019. Towards generalizable neuro-symbolic systems for commonsense question answering. In *Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing*, pages 22–32, Hong Kong, China. Association for Computational Linguistics.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2016. Conceptnet 5.5: An open multilingual graph of general knowledge. In *AAAI Conference on Artificial Intelligence*, AAAI, pages 4444–4451. arXiv.

Alexandros Vassiliades, Theodore Patkos, Vasilis Efthymiou, Antonis Bikakis, Nick Bassiliades, and Dimitris Plexousakis. 2021. *Object-Action Association Extraction from Knowledge Graphs*, pages 241–254. GmbH.

Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Kun Zhou, Yuanhang Zhou, Wayne Xin Zhao, Xiaoke Wang, and Ji-Rong Wen. 2020. Towards topic-guided conversational recommender system. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain, December 8-11, 2020*.

Yilun Zhou, Steven Schockaert, and Julie Shah. 2019. Predicting ConceptNet path quality using crowdsourced assessments of naturalness. In *The World Wide Web Conference on - WWW '19*. ACM Press.

Acknowledgements

We would like to express our gratitute to Reza, our project mentor, for his insightful comments. We also wish to thank Prof. Min-Yen for his advice and guidance.

Statement of Independent Work

You **must** *include the text of the two statements below in your group's submitted work. Digitally sign your submission using your Student Numbers (starting with A...; N.B., not your NUSNET email identifier). This is a required section and is not part of the main body (doesn't count towards your page limit).*

1A. Declaration of Original Work. By entering our Student IDs below, we certify that we completed our assignment independently of all others (except where sanctioned during in-class sessions), obeying the class policy outlined in the introductory lecture. In particular, we are allowed to discuss the problems and solutions in this assignment, but have waited at least 30 minutes by doing other activities unrelated to class before attempting to complete or modify our answers as per the class policy.

Signed, [A0194568L, A0194567M, A0211301J, A0149790R]

A Vertex and Edge Features Inherited from ZSS

Vertex features from ZSS include:

1. **Embedding.** The 300-dimensional GloVe embedding.
2. **Frequency.** Estimated frequency by taking the inverse of word rank retrieved from GloVe embeddings (by Zipf's law).
3. **Degree.** Number of words directly connected to the word in ConceptNet.
4. **Sense score.** Semantic similarity of the word with its neighbors on the path (Chen and Liu, 2011).

All the above features are pre-calculated by ZSS, so that we use them directly in our training.

Edge features from ZSS include:

1. **Ends similarity.** Cosine similarity of embeddings of the two ends of the edge.
2. **Direction.** One-hot encoded representation of the edge direction (forward, backward, or bidirectional).
3. **Relation.** One-hot encoded relation type (46 in total).
4. **Provenance.** Edge weights by 6 different provenances in ConceptNet (0 if non-exist from that provenance).
5. **Sense score.** Consistency of meaning at edge level (Chen and Liu, 2011).