# EECS 595 Final Report

**Anshul Aggarwal**
aanshul@umich.edu

**Drake Svoboda**
drakes@umich.edu

**June Park**
junkeunp@umich.edu

## 1 Introduction

In this report we present our approaches to all three tasks. Section 2 covers CommonsenseQA, section 3 covers conversational entailment, and section 4 covers EAT.

## 2 CommonsenseQA

CommonsenseQA (Talmor et al., 2018) is a question-answering dataset of 12,102 questions, each with 5 multiple-choice answers. The task is to predict the correct answer given an input question and multiple-choice answers. We tried two approaches for this task. The first was to fine-tune a BERT transformer model. The second is a novel graph neural network (GNN) based approach that leverages knowledge from ConceptNet (Liu and Singh, 2004). Since CommonsenseQA is derived from ConceptNet, we believe it would be beneficial to include commonsense knowledge in the input representation. The BERT transformer outperformed the GNN by a large margin. This is not surprising considering that BERT is a vast model that benefits from pre-training on a large corpus.

In this section we will briefly describe our BERT implementation. We will then thoroughly describe our GNN-based approach.

### 2.1 Bert Transformer

Our first approach is to fine-tune a BERT transformer. We choose `bert-base-uncased` from the Hugging Face library as our BERT encoder and used their implementation for `BERTForMultipleChoice`.

The BERT transformer is pre-trained for the masked language modeling (MLM) and next sentence prediction (NSP) tasks on a large corpus of books and Wikipedia articles. The pre-trained model is fitted with a randomly initialized classification head before fine-tuning.

The model is trained for 3 epochs with a batch size of 4 (7308 iterations) using the AdamW optimizer with an L2 penalty with $\lambda = 0.01$. The L2 penalty is not applied to layer norm or bias parameters. During the first 500 iterations of training, the learning rate is linearly increased from 0 to $5e-5$. The learning rate is then decreased back to 0 for the remaining training iterations. At each iteration, the computed gradient is clipped to have a maximum norm of 1. This training procedure is default for the Hugging Face `Trainer` class. After training, we achieve a validation accuracy of 56.59%. We tried other training parameters, however none improved upon the defaults provided by Hugging Face.

### 2.2 Graph Neural Networks

For our second approach we use graph neural networks (GNNs). GNNs operate on graphs. Each layer in a GNN uses a *message passing* scheme that updates the hidden state of a vertex by aggregating information from adjacent vertices in the graph. A simple formula for message passing is as follows (though there exists more general formulations):

$$h_i^{k+1} = \sigma(U^k h_i^k + \sum_{j \in N(i)} (V^k h_j^k)) \qquad (1)$$

where $h_i^k$ is a the hidden vector representation for vertex $i$ at layer $k$, $N(i)$ is the neighborhood around vertex $i$, $U^k$ and $V^k$ are learnable weight matrices for layer $k$, and $\sigma$ is some non-linear function like ReLU.

A transformer's *attention* mechanism is mathematically similar to message passing and is defined as follows:

$$h_i^{k+1} = \sum_{j \in S} w_{ij}(V^k h_j^k)$$

where

$$w_{ij} = \text{softmax}_j(Q^k h_i^k \cdot K^l h_j^k)$$

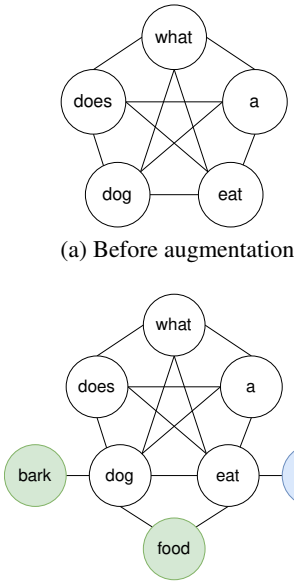
$S$ is the set of tokens in the sequence, $V$, $Q$, and $K$ are learnable weight matricies, softmax$_j$ is the softmax function over each token $j$, and $\cdot$ represents the dot product. At each layer in a transformer, the hidden state for each token is updated via a weighted aggregation of the hidden states of the other tokens in the sequence. Thus, we can think of a transformer as a GNN that operates on a fully connected graph of tokens. This leads to the question: is a fully connected graph of tokens the best representation for an input sequence? We think that there is room to improve upon this representation.

CommonsenseQA relies on prior commonsense knowledge; we can encode commonsense knowledge into the input representation by augmenting the fully connected graph representation with vertices and edges extracted from ConceptNet. To produce the augmented graph, we iterate each token in the sequence, extract each of it's connected concepts, and add them to the graph. To ensure that each of our multiple choice options are in the graph, we similarly add them as vertices and also add each of their connected concepts. This procedure quickly increases the size of our graph. To reduce our representation to a manageable size, we remove all vertices that only have a single neighbor. Figure 1 shows an augmented graph representation for a simple example.

To pass through our model, each vertex is mapped to a learnable embedding in $\mathbb{R}^d$. We might choose the full set of ConceptNet concepts as our vocabulary to embed each vertex; however, ConceptNet has 1.8 million concepts and this would likely produce too large of a vocabulary. We could also try using BERT's vocabulary to embed each vertex. Many of the concepts from concept net are compound words like "revolving door" or short phrases like "going to opera"; these concepts do not exist in BERT's vocabulary but may provide important information. Instead, we choose to augment BERT's vocabulary by adding each concept from ConceptNet that is connected to at least 3 tokens in BERT's default vocabulary. This yields a vocabulary of size 220,746. Using this vocabulary we can embed each vertex to a fixed length vector. Like BERT, the embeddings also include information regarding the token's type and position:

$$h_i^1 = E_i + P_i + T_i$$

where $h_i^1$ is the embedding for vertex $i$, $E_i$ is the embedding for $i$'s token, $P_i$ is the embedding for $i$'s position (the position embedding is the zero vector if the vertex is a concept not apart of the input sequence), and $T_i$ is the embedding for $i$'s type. We use a set of three possible vertex types: *Sentence* if the vertex belongs to the input sequence, *Concept* if the vertex is an added concept, and *Answer* if the vertex is a multiple choice answer.



(a) Before augmentation

(b) After augmentation

Figure 1: Graph representations for the input question "what does a dog eat" and two multiple choice answers "bark" and "food." Each additional edge in the second graph is added from ConceptNet. Blue vertices correspond to concepts in ConceptNet. Green vertices correspond to multiple choice answers. This is a limited example, in reality the augmented graph would have many more vertices. Edge type labels are omitted.

Beyond embedding each vertex, we also embed each edge in the graph. ConceptNet has various relation types: */r/IsA*, */r/PartOf*, and */r/HasA* are examples. Thus, relations in ConceptNet can be of the form "*team* has a *coach*" or "*student* is a *person*." These edge types provide important contextual information in the graph and are retained in our representation. Edges connecting the vertices from the original sequence are given the type */r/Sentence*. Like the verticies, each edge is mapped to a learnable embedding in $\mathbb{R}^d$; rather than use the message passing formulation in equation 1, we use a more general form that uses edge information during aggregation (Li et al., 2020).

The embedded graph representation is passed through a GNN encoder that genrates a hidden feature representation for each vertex. Then, we take the feature representations for each vertex that

corresponds to a multiple choice answer and pass it through a densely connected predictor that produces a single value. The final prediction is then the softmax over these values.

The specific architecture used in our experiments is made up of 8 message passing layers. The vertex and edge embeddings each have 64 dimensions. Each message passing layer is followed by layer norm and a GELU activation function. The densly connected predictor has two linear layers separated by a GELU.

**Transfer Learning** Transfer learning has been a huge breakthrough in NLP. We think transfer learning would also be beneficial for GNNs. We pre-train our GNN on the masked language modeling task. We select CommonGen (Lin et al., 2019)—a corpus of 50 thousand simple sentences—for pre-training. Each token in each sentence is randomly selected for masking with probability $0.15$. We then construct the input graph as before. The model is then tasked with predicting the original token for every masked vertex. We train using the AdamW optimizer and a fixed learning rate of $1e-3$. After 41 epochs of training the model is able to correctly predict the masked token about 37% of the time on a held out validation set.

**Experiments & Evaluation** We train on CommonsenseQA for 3 epochs using a batch size of 64. Again we use the AdamW optimizer and a fixed learning rate of $1e-3$. We apply the L2 penalty with $\lambda = 0.01$. We achieve a validation accuracy of 28.58% when staring from a randomly initialized model. Surprisingly, using the pre-trained model was detrimental to performance and only achieved an accuracy 25.96%.

Since each question has 5 multiple choice answers, chance performance is 20%. Thus, both models perform better than chance. One of the choices is a human-generated *distractor* answer not sampled from ConceptNet. Eliminating this choice would put chance performance at 25%. It is possible that our input representation makes it easy for the model to eliminate the distractor and the model only performs marginally better than chance prediction.

**Issues & Future Work.** Our GNN model did not perform that well. This is in part because of the limited number of training examples in CommonsenseQA. Although our pre-training experiment did not improve performance, we still believe that pre-training on a larger scale would be beneficial.

Many ad-hoc decisions were made when constructing the vocabulary and model architecture. In the future we would like to try different architectures and message passing formulations; graph attention (Veličković et al., 2018) might be a good fit since it is are more similar to transformer attention. Also, there is likely a better way to embed compound-word and phrase concepts like "revolving door" or "going to opera" rather than adding them to the vocabulary. For example, it might be better to tokenize the concept first using the default BERT tokenizer, then take the average embedding among all tokens in the compound-word or phrase. Currently, 17.36% of questions in CommonsenseQA have an answer that does not belong to our vocabulary. A better scheme for embedding concepts could alleviate this problem.

We would also like to try easier NLP tasks first before language understanding tasks. We achieve good performance for MLM which suggests that our approach may be viable for other tasks.

We should also conduct an ablation study where we only use the fully connected graph of tokens from the input sequence; this would help in determining the value of adding concepts to the input graph.

Overall we are happy with the GNN's performance, but there is a lot of room for improvement.

## 3 Conversational Entailment

Conversational entailment is an entailment task where the premise is a dialog between two speakers. In this section we describe two approaches for conversational entailment. In the first approach we use SBERT (Reimers and Gurevych, 2019). SBERT is a Siamese network that leverages BERT transformers. In the second approach we fine-tune RoBERTa. RoBERTa performs best among the two approaches. We experience difficulties getting SBERT to consistently converge. We also find that it is crucial to encode speaker information in the model's input.

### 3.1 SBERT

SBERT is a Siamese network that separately encodes two sequences using two BERT transformers. The encoded sequences are joined together and used as input for a softmax classifier that makes a prediction. The structure of the model is shown in figure 2. We use SBERT for entailment by having the first BERT transformer encodes the premise

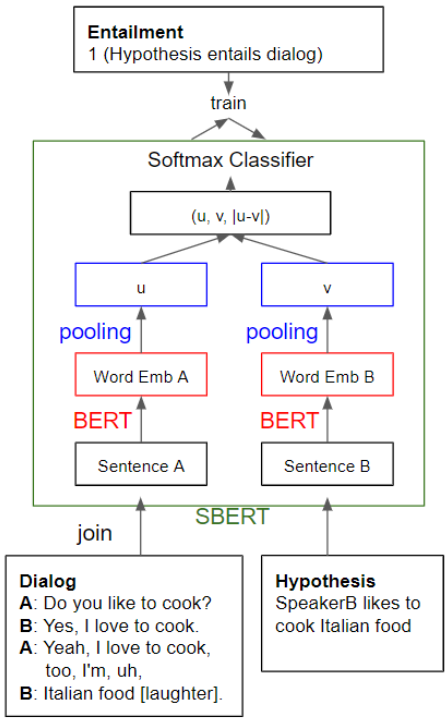



Figure 2: SBERT. The input dialog is concatenated and encoded by the transformer on the left. The hypothesis is encoded by the transformer on the right. Both transformers share the same weights.

and the second BERT transformer encodes the hypothesis. Both BERT encodings are pooled to fixed length embeddings. The embeddings are joined and used as input for a softmax classifier which predicts an entailment label. To prepare a dialog for SBERT, we concatenated all utterances into a single sequence to use as the premise. The hypothesis remains unchanged. This input representation does not include any information about the speakers.

**Results** To evaluate, we split the dataset into training and validation sets. We were unable to get SBERT to consistently converge. The validation accuracies ranged from 55% to 63% depending on the specific training and validation split. This is not bad performance, however, in many experiments the models performance failed to improve while training. We think that to get better performance with this model, we should encode speaker information in the input and more exhaustively search for good hyper-parameters.

### 3.2 RoBERTa

For our second approach we fine-tune RoBERTa. We choose `roberta-large-mnli` from the Hugging Face library. This model is pre-trained on the MultiNLI textual entailment corpus (Williams et al., 2018). This pre-training task is well suited for transfer learning on other natural language understanding tasks. We rely on Hugging Face's implementation for `RobertaForSequenceClassification`. The input to the model is created by concatenating each turn in the input dialog. We include speaker information directly by converting each dialog turn into the following format: "Speaker[A/B] said '[dialog turn]"'. The hypothesis is added at the end of the sequence after a separator token. For example the dialog in figure 3 would be converted to the following input sequence:

> SpeakerA said "Do you like to read?" SpeakerB said "Yes I do" ⟨/s⟩ SpeakerA likes to read

The model is tasked with making a binary entailment prediction on the whole sequence.

**Experiments & Evaluation** We train the model for 8 epochs with a batch size of 6 (608 iterations). We use the AdamW optimizer and an L2 penalty with $\lambda = 0.01$. The L2 penalty is not applied to bias or layer norm parameters. We use different learning rates for the RoBERTa encoder and the classification head. For the first 2 epochs the learning rates are linearly increased from 0 to $1e-4$ and $1e-8$ for the classification head and encoder respectively. The learning rates are then linearly decreased to zero for the remaining iterations. The intuition for using a smaller learning rate on the encoder is that the encoder starts with a good pre-trained initialization; thus, the weights in the encoder should be able to adapt to new tasks with smaller updates than the final classification layers. To evaulate our model, we perform 3-fold cross-validation and record accuracy for each fold. Our model achieves a macro average accuracy across the 3 folds of 74.05%.

Including the speaker information in the input is crucial for performance. Without speaker information (simply concatenating dialog turns), the model only achieves a macro average accuracy of 67.70%. This is not surprising. Consider the example in figure 3. In this example the hypothesis does not entail the dialog since speaker A does not say if she likes to read. This example would be impossible to distinguish without speaker information. Other examples in the corpus are similarly undecidable

A: Do you like to read?
B: Yes I do.
Hypothesis: SpeakerA likes to read

Figure 3: Example conversational entailment input.

without contextual information about the speakers. For future work, there is likely more clever ways of encoding speaker information rather than augmenting the input.

## 4 EAT (Everyday Actions in Text)

The EAT dataset consists of 1044 short stories, labelled as plausible and implausible. The implausible stories have further labels which indicate when the stories stop making sense. The task at hand is to predict the label and breakpoint for stories not previously seen by the model.

We randomly split the dataset into training and validation sets of 944 and 100 stories respectively.

### 4.1 First Attempt at Baseline Models

Our first approach is to split the task into 2 independent sub-problems—first to train a binary classifier which predicts if a given story makes sense or not, followed by a multi-class classifier which predicts the breakpoint for stories deemed as implausible by the first model.

We trained many different models for the first task. We implemented many sequence classification models with various pretrained configurations of BERT variants, and trained a classification layer on top of these models to fine tune to our task. Some of the pretrained models we used were `bert-base-uncased`, `xlnet-base-cased` and `roberta-base` among others. We simply took the stories as they were, adding the appropriate classification label and concatenating them with the appropriate separator tokens according to the pretrained model.

For the second task of predicting breakpoints, we trained similar models as above. The dataset consisted of only implausible stories, with the breakpoint as ground truths.

However, the best accuracy for the first tasks was a little under 60% and for breakpoint prediction was a little over 30%. Both models were performing only slightly better than chance, and this was not acceptable. We needed high accuracy in label prediction for the breakpoint prediction to make any sense at all, and a pipeline with such low-accuracy models was bound to perform poorly. We decided to abandon this approach.

### 4.2 Digging into the Dataset

After the first failed attempt at building a decent baseline model, we decided to take a closer look at the dataset. Looking at figure 4 it is clear that the dataset is heavily skewed, which partly explains why our first approach failed. Although the labels are evenly distributed, the breakpoints within the implausible stories are not, and we think the model failed to capture this trend.

Taking a closer look at some of the stories, we see a lot of repeated stories in which just one sentence is changed, which changes the semantics of the sentence and make the story implausible. For example, here are two such stories:

**Plausible story**:

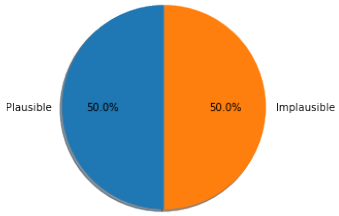
0. Ann sat down on the couch.
1. Ann reached for the cellphone on the table.
2. Ann knocked her coffee off the table.
3. Ann got some cleaning supplies to clean up the mess.
4. Ann scrubbed at the coffee stain.

**Implausible story**:

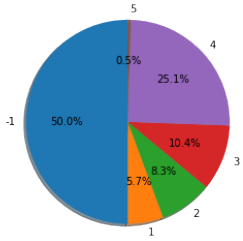
0. Ann sat down on the couch.
1. Ann reached for the cellphone on the table.
2. Ann knocked her coffee off the table.
3. Ann got some cleaning supplies to clean up the mess.
4. Ann drank her coffee.

Most stories had such plausible and implausible counterparts. The implausible story in the above example has a breakpoint at sentence 4, however the challenging aspect is the skewed distribution of the breakpoints among implausible stories.

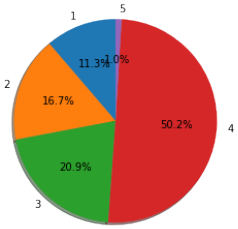
Our new intuition is that for any model to accurately predict plausibility, it must be able to implicitly predict the breakpoint. This gives us a tight dependency between the two seemingly independent tasks and forces us to rethink our approach to the problem. We can no longer try the pipeline approach of first predicting plausibility. Our main takeaways from this analysis are:




(a) Distribution of Labels in Training Data




(b) Distribution of Breakpoints in Training Data




(c) Distribution of Breakpoints in Implausible Stories

Figure 4: Label and breakpoint distributions for EAT.

1. Do not underestimate the inherent complexity of the task

2. The EAT dataset has very few examples and many examples are very similar

3. Blindly applying BERT variants does not give meaningful results

4. Both sub-tasks are tightly dependent

5. Approaches predicting labels before breakpoints are bound to fail

### 4.3 A Better Approach

We were desperate.
We needed more data.
We needed a working model.

We flipped our first approach.

A closer look at the dataset reveals that even for implausible stories, the story formed before the breakpoint is plausible (and this is true for all sub-stories before the breakpoint). Armed with this revelation, we split each story into many sub-stories. A plausible story, say of length 5, is split into 4 plausible sub-stories. Length one stories are not included as they are never implausible. For implausible stories, we try two approaches. For the first approach, each sub-story is created and all stories which contain the breakpoint are marked implausible. For the second approach, we discard sub-stories that include sentences after the breakpoint. This increases the size of the dataset by almost 3-4 times, although the lengths of the implausible stories are still biased towards longer stories (which follows from the breakpoint distribution).

We then train a model to predict if the last sentence in an input story aligns with the prior; this is similar to textual entailment. We predict the breakpoint as the first non-sense sentence that is found, and report no breakpoint (-1) if no non-sense sentence is found. The new approach converts the multi-class breakpoint classification into a binary classification and eliminates the need to predict plausibility of the complete story.

We train `bert-base-uncased` for this task and achieve 52% accuracy (macro F-1 score) for breakpoint prediction (much better than chance). The confusion matrix in figure 5 sums up our results.

While this model shows promise, it comes with it's own baggage. This model is cheating.

1. The model is biased towards predicting no breakpoint, possibly because our story-splitting approach introduces more bias towards plausible stories

2. The model predicts no-breakpoint more often for stories with a breakpoint at 3 or 4

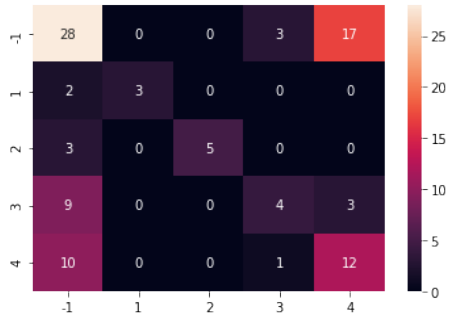



Figure 5: Breakpoint prediction confusion matrix for `bert-base-uncased`. The y axis shows the ground truth breakpoint. The x axis is the models prediction. -1 corresponds to no breakpoint (story is plausible).

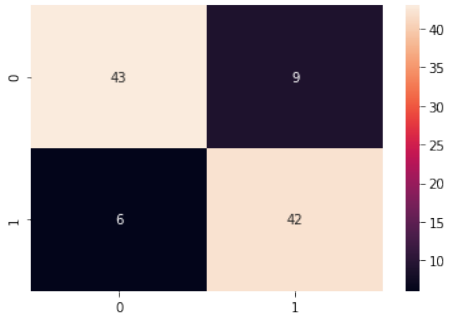



Figure 6: Plausibility prediction confusion matrix for RoBERTa. The y axis shows the ground truth label. The x axis is the models prediction. Label 0 corresponds to implausible, 1 is plausible.

3. For a large number of plausible stories, the model predicts the breakpoint at sentence 4—essentially performing worse than chance for breakpoint 4

Although we were expecting points 1 and 2, point 3 took us by surprise, especially because breakpoint 4 is the most common breakpoint—and we just cannot afford such a poor performance on breakpoint 4.

### 4.4 Enter RoBERTa + MNLI

The next natural progression is to try bigger and better pretrained models, especially focusing on models trained on other language understanding tasks.

We try `bert-base-mnli`, Facebook's `bart-large-mnli` (Lewis et al., 2019) and Hugging Face's `roberta-large-mnli`. While the BERT and the BART variants tanked, RoBERTa showed a significant improvement. Figures 6 and 7 show RoBERTa's performance. Figure 7 shows a considerable improvement over the BERT baseline in figure 5. RoBERTa is not overly biased towards predicting no-breakpoint, performs well for breakpoint 4, and also predicts breakpoint 5 correctly (only a few examples in the dataset have breakpoint 5, breakpoint 5 is not included in the confusion matrix). These results show that the model is no longer cheating and is able to accurately predict breakpoints for different length stories.

| Score | Label | Breakpoint |
|---|---|---|
| F-1 | 0.9 | 0.8893 |
| Recall | 0.9014 | 0.8714 |
| Precision | 0.9014 | 0.9173 |

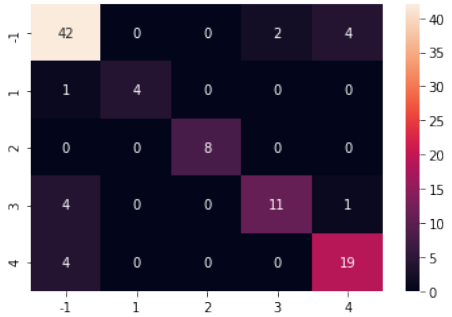



Figure 7: Breakpoint prediction confusion matrix for RoBERTa. The y axis shows the ground truth breakpoint. The x axis is the models prediction. -1 corresponds to no breakpoint (story is plausible).

## 5 Conclusion

Overall, we were able to achieve good performance on all three tasks; pre-trained transformers proved to be superior in all cases. This is not surprising since they are the current state-of-the-art in NLP. We have learned that blindly applying transformers will not solve all NLP problems; instead, it is important to carefully consider the dataset and approach.

## References

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns.

Bill Yuchen Lin, Ming Shen, Wangchunshu Zhou, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang

Ren. 2019. Commongen: A constrained text generation challenge for generative commonsense reasoning. *CoRR*, abs/1911.03705.

H. Liu and P. Singh. 2004. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *CoRR*, abs/1811.00937.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.