Extending BERT for multi-choice problems

Orr Dermer [*1] **Afek Adler** [*2]

Abstract

Deep learning models are still far away from human-level performance in commonsense reasoning tasks. We introduce a multi-choice question answering scheme that models the dependencies between answers by building a relationship graph between the answers for a given question. Then, we compute a centrality measure for each node (answer) in the graph and predict the answer by taking the node with the maximum value of that centrality measure. We test ourselves on the challenging CommonsenseQA dataset and show that although our method is capable of modeling more complex relationships between the answers we did not manage to improve the current state of the art on this task.

1. Introduction

When we face a multi-choice problem where we have to choose the most probable answer we arrive to it with prior information and knowledge. e.g, example, we have some kind of a language model that tells us if sentence A is more probable then sentence B, and we have knowledge about the physical world which tells us what is more possible or probable.

While in some cases the state of the art machine-learning methods which were trained on massive amount of data to infer prediction roles outperforms human-level by a large margin (e.g. SQUAD 2.0 (Rajpurkar et al., 2018)), In the field of common sense reasoning (Talmor et al., 2018) state of the art algorithms fail to generalize and are yet inferior to human level performance. This is due to two main reasons: first, is that when using algorithms such as fine-tuning a pre-trained language model, it is likely that this knowledge (common sense) does not exist explicitly in the pre-training corpus. And second, is due to the fact that creating massive datasets for this purpose solely is hard, and developed only in the last years mostly by crowdsourcing.

Due to this inherent difficulty of this task, the research community invests resources in creating larger-scale datasets with different difficulty levels, e.g., the SWAG (Zellers et al., 2018) dataset with 100K examples for which (Devlin et al., 2018) demonstrate a model which is already at a human level, and the CommonseneQA (Talmor et al., 2018), a dataset which was made especially for common sense reasoning with ˜10K samples where the state of the art is much lower then human-level. For a detailed survey of resources see (Storks et al., 2019).

In this work, we suggest and evaluate models which eliminate the Independence assumption between answers for a given question, aiming to improve the benchmark on the CommonsenseQA dataset.

2. Background and Related Work

Research on machine common sense QA has long been acknowledged for its critical component in natural language understanding. However, only recently, with the embracing of deep learning in natural language processing and crowdsourcing to create larger scale datasets, major breakthroughs has emerged; To the best of the author's knowledge, the first large scale common sense reasoning QA dataset was introduced is 2018 (Storks et al., 2019).

Due to the fact that the prominent approach to solve common sense QA problems is with word embeddings (WE) and contextualized word embeddings (CWE) in the next section we will provide a brief introduction to CWE and explain how to use it in the context of common sense QA.

2.1. Contextualized Word Embeddings

Based on the pioneering work of Word2Vec (Mikolov et al., 2013) which enabled words to be represented as vectors, thus enriching the information contained in the input for a variation of language based machine learning tasks, many models extended the Word2Vec concept by taking context into account; the same word has a different meaning in different context. The leading approaches to obtain a contextualized word representations is using the hidden layer

[*]Equal contribution [1]Department of Computer Science, University of Tel Aviv, Tel Aviv, Israel. [2]Department of Industrial Engineering, University of Tel Aviv, Tel Aviv, Israel. Correspondence to: Afek Adler <afekilayadler@gmail.com>, Orr Dermer <odermer@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

of an LSTM (Hochreiter & Schmidhuber, 1997) and more specifically a Bi-LSTM such as in ELMO (Peters et al., 2018) or more recently using a transformer architecture such as BERT (Devlin et al., 2018; Yang et al., 2019).

The great power of WE or CWE is to enable using a generic method as a basis for task specific machine learning models; For instance, in the field of natural language processing there are many different tasks (e.g. machine translation, sentiment analysis), yet a simple model based on CWE with fine tuning on a given dataset usually provides much better results than previous models on a given task, creating a new benchmark for this specific task. Companies interested in the field of natural language processing often release those trained models as an open source service, which makes CWE easy to use for academics and in the industry as one.

CWE is a type of a transfer learning method; a model is pretrained on a base dataset and fine tuned on the specific task dataset, with the underlying assumption that information distillation is possible from one source (base dataset) to the other (task specific dataset). The main reason for the great success of CWE is that the base model is trained using supervised learning on huge amounts of data, e.g. BERT, was trained on 3.3B words. The fact that CWE models are also self trained makes it easier to collect such huge amount of data without the need to label samples. The downside of these models is that they use many parameters (BERT Large - 340M) - so much such that training and infering requires large processing power and latency is a challenge for creating real time systems. In the next section we will introduce the benchmark model which we aim to improve - BERT for CommonsenseQA (which is identical to BERT for SWAG) as was introduced in the original paper of BERT.

2.2. BERT

BERT is a word based CWE model that was pre-trained on two learning objectives: Masked word predictions (15% of the words were masked) and next sentence predictions. BERT's model architecture is a multi-layer bidirectional Transformer encoder (**?**) with the following hyperparameters (BERT Large) - 12 Transformer blocks, hidden size 1024 (H) and 16 self attention heads. BERT takes as input one or two sentences.

BERT's input representation (figure 1) uses WordPeace (Wu et al., 2016) tokenization which is a data driven method that aims to achieve a balance between vocabulary size and out-of-vocabulary words. This way BERT stores only 30K vocabulary words, yet very rarely encounters out-of-vocabulary words when tokenizing English texts. Two special token are added for each sentence, a start token [CLS] and an end token [SEP]. If the input contains two sentences another [SEP] token is added in between the sentences. Due to the fact that Transformers do not encode the sequential nature of their inputs a segment ids and position embeddings are added as well to the input.

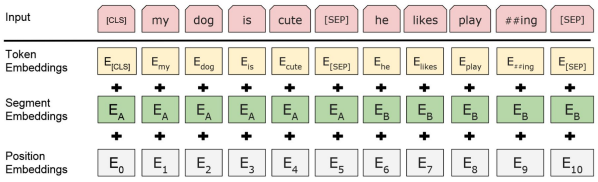

Figure 1. input representation for BERT (from the original paper).

Fine tuning BERT is performed by appending the task specific classification layer with a given loss function to the pretrained model, training using the specific task's input / output, and changing parameters using back-propagation.

2.3. BERT baseline for SWAG

In the BERT Paper, the authors approach the SWAG challange with the described input (figure 2). that input is fed to BERT's architecture and the [CLS] token embedding is used to predict a logit by calculating a dot product with a learned vector. this process is done for each answer independently of each other and then a softmax layer with cross entropy loss is calculated to compute the error. As we can see, this architecture introduces only H (1024) new parameters to the existing BERT model.

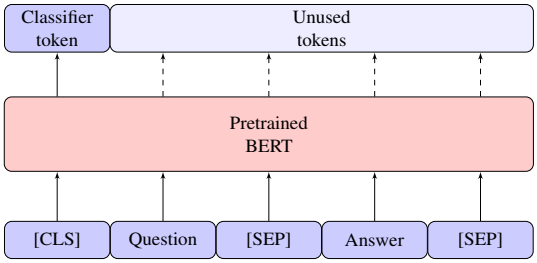

Figure 2. The original paper's question-answering model.

3. CommonSeneseQA

CommonSenseQA is a dataset specifically made for common sense reasoning, by asking crowd workers to generate questions from concepts from ConceptNet. It contains 12,247 examples and 5 answers per question. the Authors use the same architecture as in the BERT benchmark for SWAG and report an accuracy score of 56.7%. The current state of the art on this dataset is currently 68.3% accuracy, which is 23.6% below human level.

4. Methodology

We propose a method with different variants to model the dependencies between answers, by feeding answer pairs jointly to BERT's architecture. Doing so, we contribute by:

- making the model more expressive by removing the independence assumption
- extending the model to deal with questions as 'what is largest?"

We derive our model from the BERT baseline for SWAG. However, as we want our model to compare between the different possible answers, we deviate from the original model by introducing the model with the question and 2 possible answers as each input sequence, for which the model returns a score - 1 meaning the first answer is more likely, and 0 if the second answer is the likeliest. Having received a score for each of pairs, we then integrate those into probability scores for each of the original answers - and then choose the one with the highest probability as our predicted answer. A full overview of our model can be seen in figure 3.

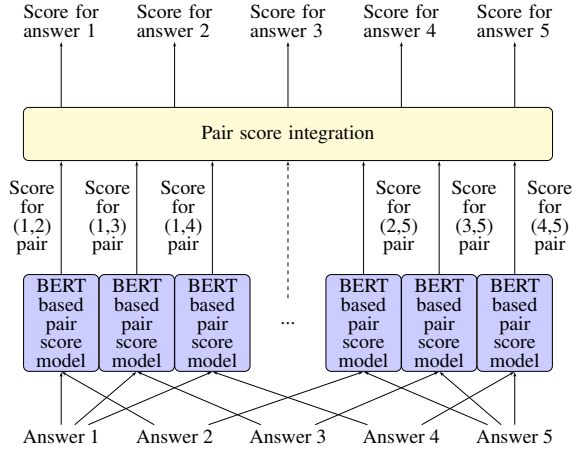

Figure 3. Our full model overview.

4.1. Dataset alterations

In the original paper, each input sequence consisted of a classifier token ('[CLS]'), the question, a separator token ('[SEP]'), a possible answer, and then another separator token, as seen in figure 2. The classifier token is unique, as the final hidden state corresponding to that token was used as the aggregate sequence representation for the question-answering task.

Our model deviates from these assumptions. Instead, each input sequence was constructed from the question and 2 possible answers in the following manner: A classifier token, the first answer, a separator token, the question, a separator token, the second answer, and another classifier token (As seen in figure 4). Note, that this construction breaks several of the original BERT paper assumptions - each of our input sequences consist of more than just one classifier token, and more than just two sentences. As such, it is possible that BERT will suffer from reduced performance, as its entire training process was built around these assumptions. However, breaking these assumptions was a crucial part of what made our constructions work - it allowed for each of the classifier tokens to "absorb" the meaning of the answer closer to it, and imporved the symmetry of our model in regards to each of the two answers.

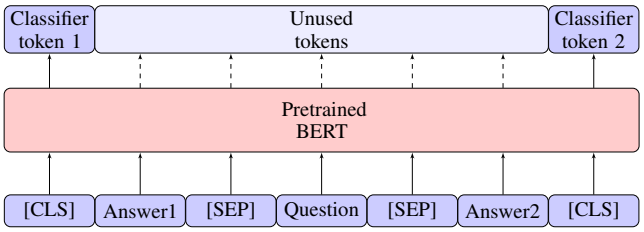

Figure 4. Our question-answering model.

Another part of the input sequence given to BERT is the segment embeddings - a number from $\{0, 1\}$ which signifies whether the current token is part of the first sentence or the second. After trial and error, it was found that we perform best when giving a value of 1 for **each** of the answer tokens, and 0 for the question tokens. That makes sense, as part of the desire for symmetry in our model, and the fact that BERT wasn't trained with a possibility for a third sentence, so using, for example, the number 2 for our second answer would not make any sense to it.

After passing our input sequence through BERT, we use a concatenation of the final hidden states matching the classifier tokens as the sequence representation. We learn as a parameter a vector whose dot product with the sequence representation is the score for the given answer pair (equation 1). Then, we compute the edge score between answers i to j by averaging the sigmoid result of the two.

$$\begin{aligned} C\vec{T}_{i,j} &:= \textit{Classifier Tokens Concatenated}\ ^{1} \\ \vec{P} &:= \textit{Learned parameters} \\ S'_{i,j} &:= sigmoid(C\vec{T}_{i,j} \cdot \vec{P}) \\ Score_{i,j} &:= \frac{S'_{i,j} + (1 - S'_{j,i}))}{2} \end{aligned} \tag{1}$$

[1]Classifier tokens concatenated, after running through the BERT model using answers i, j for input

4.2. Pair scores integration

After runnning the previously described part of the model, we receive a score for each of the answer pairs, meaning "How much more likely is answer A than answer B". We conceived a few methods of combining the different pair scores into probabilities for the answers. In this section we will review those methods and compare them.

4.2.1. Learnt weights

In this method, we introduce more layers to derive the indiviual probabilities from the pair scores. We use 2 layers, with a hidden size of $(number\ of\ pairs)^2$ and normalizing with a softmax layer. This generates a valid probability distribution over the possible answers.

4.2.2. Markov chains

In this method, we find the stationary distribution of a weighted graph whose edge's weights are derived from our pair scores. The reasoning for this process is as follows - imagine a graph whose vertices are the given answers. We begin by choosing an answer randomly, and we want to move on to the best possible answer. At each time step, we move to a different answer, choosing based on the scores of the pairs in which our current answer participates, i.e. the better another answer performed against our current answer, the more likely we are to move to it. This process is powerful, because it captures the dependency between the different answers, e.g. if answer B is better than answer A, and answer C is better than answer B, then we'd like to factor in that information when comparing between answers A and C.

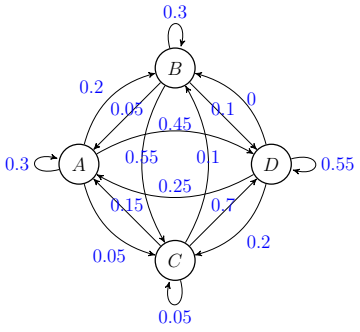

Figure 5. A markov chain with 4 nodes.

Simply constructing a matrix with each cell being the score for the matching pair, normalizing each row so that we have a probability matrix, and then calculating the stationary distribution doesn't work well in practice. This is because normalizing the matrix causes the model to be unable to capture the absolute value of the scores, e.g. if it sees a row in which all values are similar, it can't differ the case in which all scores were low (and thus our "current" answer is good) from the case all of the scores were high. Consider the following pair-score matrix, extracted from a sample run of our previously discussed model:

$$\begin{bmatrix} 0 & 0.95 & 0.85 & 0.25 \\ 0.05 & 0 & 0.98 & 0.74 \\ 0.15 & 0.02 & 0 & 0.45 \\ 0.75 & 0.26 & 0.55 & 0 \end{bmatrix}$$

We can see that the third answer had won in every comparison (every non-diagonal cell in the third column is > 0.5) - thus, we would expect it to be assigned the highest probability. Given the currently described method, this in not the case, as the probabilities received would be:

$$\begin{bmatrix} 0.22614831 & 0.16630443 & 0.29576885 & \mathbf{0.31177841} \end{bmatrix}$$

We solve this problem by adding the average of each column to the matching cell on the diagonal. This makes sense, as this gives us the option of staying on the current choice, and the better a certain choice had performed against the other choices, the greater the possibility of staying on it. After doing this, we derive this matrix for our example:

$$\begin{bmatrix} \mathbf{0.31} & 0.95 & 0.85 & 0.25 \\ 0.05 & \mathbf{0.41} & 0.98 & 0.74 \\ 0.15 & 0.02 & \mathbf{0.79} & 0.45 \\ 0.75 & 0.26 & 0.55 & \mathbf{0.48} \end{bmatrix}$$

We then normalize the matrix and calculate the stationary probability distribution, which now seems to describe the relationships between our answers well:

$$\begin{bmatrix} 0.16845236 & 0.13252954 & \mathbf{0.43521662} & 0.26380148 \end{bmatrix}$$

4.2.3. Tie-breakers

This is not a method on its own, but it can be used on top of the other ones. Essentially, we can assert a "Certainty Threshold" on the probabilities obtained from the other methods, a rate between two probabilities under which we consider our prediction to be uncertain/unstable. If such a scenario arises, we then resort to choosing between the two highest-scored choices directly - that is, by looking at the score given to the pair and deciding accordingly.

Note that this change does not cause a change in the probabilities themselves, it only changes the predicted answer. As such, it does not cause a change in the measured loss, thus does not affect the learned parameters. It does change, however, the measured accuracy, which means that when learning parameters we will favor a parameter set that maximizes the accuracy given this change.

5. Experiments and Results

As our model deviates greatly from the assumptions that were made when the original BERT weights were trained, it is understandable that our model required more epochs to converge than the amount of epochs the original paper used when fine-tuning to the question answering task. We should also note that because each sample now consists of much more input sequences (the original model had one sequence per possible answer, our model has one sequence per possible answer **pair**), we are now more limited in our batch size, as the GPU has to hold the entire batch simultaneously. This problem can be mitigated by using gradient accumulation.

Our models were trained in parts. In the first part, we trained solely on the dataset alterations part, without integrating the pair scores. That is, our model was trained directly on identifying the better answer from a pair of answers. After 5 epochs with a dynamic learning rate (starting from 1e-6 and decreasing to 4e-7) we were able to achieve a 70.3% accuracy on the dev set. (Note that this accuracy is measured over pairs and not over questions - it is not comparable to the performance of other models over this dataset).

In the second part, we trained our model for 2 more epochs using each method of pair-scores integration suggested. Empirically, using the tie-breakers method on any of the suggested models caused a drop in performance, and thus it was discarded. A comparison of our experiment results follows.

METHOD	ACCURACY (DEV-SET)
COMPARATIVE MODEL + MARKOV CHAINS	49.3%
COMPARATIVE MODEL + LEARNT WEIGHTS	49.1%
RANDOM GUESS	25%
ORIGINAL BERT MODEL (BASELINE)	**56.7%**

Table 1. Full model comparison

6. Discussion

6.1. Train-Test Discrepancy

When humans tackle multi-choice questions we compare all answer pairs simultaneously, disqualify some unreasonable answers and approve others before we make a prediction. Thus, we narrow the number of possible solutions in a multi-phase process. In this paper, this is exactly what we wanted to achieve when we compared all answers pairs altogether. One possible reason for our model's under-performance is Train Test Discrepancy; we are aiming to learn in a supervised way information which is not exactly given to us - In the training set, we know only the gold labels per each question and not the relationship between them. Therefore, there is no information about answers which are totally wrong or answers which are close to each other and can be the prediction candidates.

Also, assuming we are given 5 possible answers, in the training phase we create 8 answer pairs - the gold answer with all the other answers (for each ordering of the pairs). However, at test time we create all possible answer pairs which are 20. It appears that at training time the model learns to discriminate between the right answer and the wrong answers, A dynamic which is not necessarily the same at test time when we feed all answer pairs together. This discrepancy can be inherent in the dataset and the way it was built, e.g. it may be that the gold answer is better then all the other candidate answer but some answers are distractors which "confuse" the model at test time.

6.2. Conclusions

Neither the Markovian model nor the Learnt weights model outperformed the baseline model. In fact, there was a significant gap between our methods and the baseline results. We believe that it is due to a combination of the following reasons:

- Train-test discrepancy issue.
- BERT was trained in a different manner than our architecture. It is likely that our input representation has harmed the information distillation from the original BERT model thus harming the final result. Which input configuration is optimal - is still in open question, in the sense that information from BERT model is distilled but also answer pair information is learnt.
- Bigger datasets can help in this manner, by enabling us to generalize well input configurations that are different than BERT baseline.

7. Future Work

To solve the train-test discrepancy issue, we believe that it is possible to create a dataset with more detailed training labels, for example, ranking the answers and providing information about distractor answers. It is interesting to verify if we can achieve a better performance by removing this discrepancy.

Another possible approach to improve upon our work with a **ranked** dataset is to fully "neuralize" our Markov chain solution. For instance, one could create a model that feeds all the answer pairs jointly to BERT, compute a logit for every pair and compare it with the **ranked** ground truth, then use convolutional layers to classify the correct answer. It can also be interesting to examine the graph layer of this

network for explainability reasons.

References

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

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

Storks, S., Gao, Q., and Chai, J. Y. Commonsense reasoning for natural language understanding: A survey of benchmarks, resources, and approaches. *arXiv preprint arXiv:1904.01172*, 2019.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

Zellers, R., Bisk, Y., Schwartz, R., and Choi, Y. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.