# CommSense: A Common Sense Questions Answering Model

**Rishabh Kumar Saxena,**[*1] **David Opie,**[*1] **Anushka Dhekne**[*1]

[1]Department of Computer Science and Electrical Engineering,
University of Maryland Baltimore County,
Baltimore County, Maryland, USA
rsaxena2@umbc.edu, dopie@umbc.edu, anushkd1@umbc.edu

## Abstract

This document provides the design of our CommonsenseQA project. The CommonsenseQA project aims to train the model according to the commonsense questions available and answer those questions based on a list of possible answers choices given to us. This project includes implementations of the Machine Learning Models and records of their accuracy. We have attempted to improve the accuracy of our model by training and validating our data and providing it to the model for fine-tuning.

## Introduction

One critical aspect when thinking of artificial intelligence is Common Sense. It is what makes us truly intelligent. We expect a human to be able to answer these type of questions, which require basic knowledge and common sense, making use of observations and previous background knowledge. For example, it is common sense for us that fire is always warm or that fish swim in water. We, as humans hold intrinsic knowledge that is hard to replicate in artificial beings or machines without training them to do so.

For a machine, which does not know or understand warmth of fire or what it means or feels like, the concept of fire being warm is not a common knowledge to it. This is the premise for the motivation of our project. We want to develop a model which can answer these common sense questions accurately, just like a human would. We want our model to know the answers to these common sense questions through training, testing and validating and get accurate results through it. We will provide our model a CommensenseQA corpus (Talmor et al. 2018) of common sense questions, multiple choice answers (to be specific, 5 choices from which 1 correct answer needs to be selected by the model) as an input. This is a standard data set for common sense question answering tasks and consists of approximately **12K** questions with five choices each - one correct choice and four distracting or wrong choices. The input will be utilized by our model to learn so that when it encounters new questions which were not already in the training set, it can answer these new questions accurately.

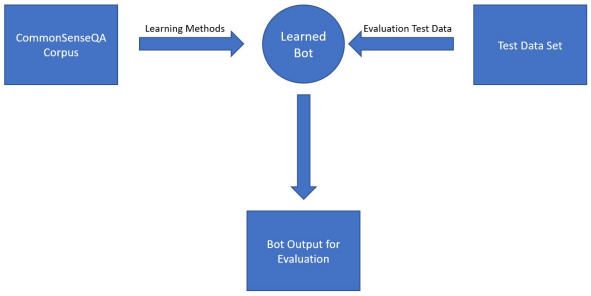


Figure 1: The overall project idea

## Related Works

### Relevant AI Techniques

There are various AI techniques currently being used in the task of linguistically understanding common sense questions and matching the question with the correct answer, given a set of candidate answers (multiple choices to select the answer from). The different natural language processing based models like BERT, Esim, GPT, and Knowledge Graphs make use of semantics and schema graphs to solve such types of problems. Making use of the libraries available for each of these models makes understanding the semantics of our language models easier and it generates a faster and more accurate mean to understand the commonsense questions and provide an answer to them from the available multiple choices.

Knowledge Graphs use abstraction for various domains where the relations between edges, paths and variables of the domain are captured through graph-based data models. These data models capture, maintain, integrate and extract data values from a large data set (Hogan et al. 2021).

GPT-3(Generative Pre-trained Transformer) language model uses deep learning in order to produce textual data like sequences of letters, words and codes from the input sent to the

[*]These authors contributed equally.

computational system. The GPT-3 language model is trained on unlabeled data sets made up of mainly English words. To get accurate and relevant output, we need to train a large set of unlabeled data for the model to be able to predict the word sequences statistically and predict answers for the questions asked (Floridi and Chiriatti 2020).

BERT(Bidirectional Encoder Representations from Transformers) uses masked language modeling, where the model randomly masks some of the input tokens. The main aim is to predict the vocabulary id of the masked data based on its context. BERT is designed to pre-train bidirectional representations from unlabeled text by jointly conditioning on both left and right context. Hence, the model can be fine tuned with an additional output layer to create models for question answering and language inference, without any task specific architecture modifications (Devlin et al. 2019).

To create our baseline accuracy, we updated existing BERT implementation on CommonsenseQA and all of its files/methods to make it compatible with the updated tensorflow version as the implementation used an old version which is not now maintained. Using the updated code we attempted to get better results by changing the hyper-parameters of our model. We tried to change the following hyper-parameters in the existing model:

- Number of Epochs
- Learning rate
- Training batch size
- Evaluation batch size
- Predict batch size

The modifications to the hyper-parameters didn't produce any noticeable results. Any changes we made made only produced an accuracy within 5 percent of our original. Hence, we implemented BERT model with ADamW optimizer where we linearly increase the learning rate until a defined number of warm up steps and linearly decrease it for the rest of the iterations. For the phase two of our project, we decided to continue with the bert-base-uncased model and improve its accuracy as compared to the accuracy of the model which we executed during phase one of the project. We provided training data, testing data and validation data to our model and made use of an additional library, deeplib. We attempted to use an application using the deeplib library, which achieved a higher model accuracy than our previous model, to improve our model but faced troubles as one model was based on tensorflow and another was based on pytorch which meant some tools were specific to each method.

## Relevant AI Literature

The current AI literature provides lots of relevant implementations and research using this data set. Given a natural language question, based on commonsense logic and a set of candidate answers to select from, the task is to select one answer from the set of options. We can select it from an external knowledge graph (Lin et al. 2019). The knowledge graph(G) can be defined from the available data as a fixed set of natural language concepts(V), with semantic relations between these concepts described by edges(E). Another work involved using the language models utilizing knowledge through pre-trained word vectors like Word2ec or contextualized word vectors using general encoded representations. Since there are only a few parameters for the model to learn from, the language models can be trained from scratch and fine-tuned later on (Mikolov et al. 2013). A semantic classifier can be trained for classification of the noisy labeled data, to create explanations for it and predict the output (Hancock et al. 2018). Language Modeling based techniques such as the GPT and BERT models can also be used for this task(Radford et al. 2018; Devlin et al. 2019).

Libraries like the 'transformers' library helps in building pre-trained models, train them and create an underlying architecture for Natural Language Understanding (NLU) and Natural Language Generation (NLG). BERT uses the 'transformers' library to process the sentences in only 2 time-steps. As transformers use non-sequential processing, therefore, the sentences which are fed to the model are processed as a single sentence, rather than processing the sentence word by word.

## Implementation Phases Details

The project was implemented in the following three phases:

- **Phase 1**: In the phase one of our project, we extended our literature review to understand more about the existing research. In this phase, we were able to implement the existing BERT model for CommonsenseQA data set. This required transitioning the model from Tensorflow version 1 to version 2. This helped us in exploring the possible hyper-parameters which could be tuned to improve accuracy of the existing model.

- **Phase 2**: During the second phase of our project, we first explored to implement GPT-3 model on CommonsenseQA dataset. We observed that it works well with open-ended questions as it can chain together low-level object details and events into a coherent explanatory structure. It means that though it works very well open-ended questions which require the continuation of response based on the context of the question but not with closed end questions(like MCQ). Hence, it could not be implemented on CommonsenseQA dataset with reasonable accuracy. Then, we focused on tuning the hyper-parameters of the existing implementation of bert-base-uncased model.

- **Phase 3**: This phase was dedicated to the write-up summarizing the implementation approach, results and conclusions of the project implementation in an AAAI formatted report.

## Libraries Used

- **OpenAI**: This open-source python library provides convenient access to the OpenAI API (which is an API used for

implementing Artificial intelligence) from applications written in the Python language. It contains pre-defined classes which are dynamically initialized from API responses.

- **Pandas**: It is also an open-source library mainly utilized to work with labeled data. It is usually used to handle and manipulate various types of data sets (csv, json, etc.)

- **Numpy**: It is a python library providing support for manipulating multi-dimensional arrays and matrices. It also provides better support for implementing mathematical operations on these arrays and matrices.

- **Tensorflow**: It is also an open source library for machine learning and artificial intelligence. It provides support of the functions to define and train the models, make predictions from them and then evaluate them based on various metrics.

- **Pytorch**: The Pytorch library is a Machine Learning Framework. This open source framework is a python programming language based framework. It is an optimized Deep Learning library. It can be used for high computational tasks which include CPUs and GPUs which also makes use of dynamic computation graphs.

- **Transformers**: The transformers library is basically a Hugging Face package which provides various pre-trained models for implementing a variety of tasks to clean, reduce, expand or generate features for model building as well as for a number of Natural Language Processing tasks. The transformers library acts on unseen data and assigns different weights to the data based on the input's significance on the output.

- **Matplotlib and Seaborn**: These open source libraries are used for visualising the results and outputs in the form of graphs, matrices, heat maps, histograms etc. They provide us with the functions to create and customise such visualizations.

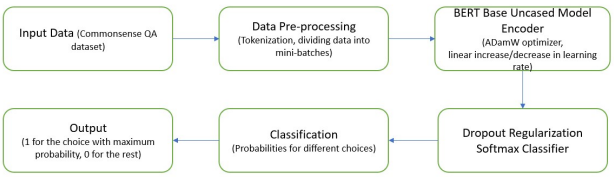


Figure 2: Implementation Approach

## Approach

The basic approach in implementation of the BERT-base-uncased model involved loading of the dataset, pre-processing of the input data, fine-tuning of the pre-trained model parameters, training the fine-tuned model and model evaluation.

- **Loading the Dataset**: CommonsenseQA is a new MCQ answering dataset that requires different types of commonsense knowledge to predict the correct answers . It contains 12,102 questions with one correct answer and four distracting answers. Metadata information and sample information is shown in Table 1 and Figure 3. The data was loaded from Huggingface's datasets library which already has validation and train split in the data.

| S.No. | Feature Name | Description |
|---|---|---|
| 1 | id | QuestionID: a unique identifier for a question |
| 2 | question | The actual question |
| 3 | question_concept | The domain of the question |
| 4 | choices | Contains Labels and textual choices |
| 5 | answerKey | The actual answer |

Table 1: CommonsenseQA Metadata


```
{
  "id": "075e483d21c29a511267ef62bedc0461",
  "question": "The sanctions against the school were a punishing blow, and they"
              "seemed to what the efforts the school had made to change?",
  "question_concept": "punishing",
  "choices": "{'label': ['A', 'B', 'C', 'D', 'E'],"
         "'text': ['ignore', 'enforce', 'authoritarian',"
           "'yell at', 'avoid']}",
  "answerKey": "A"
}
```


Figure 3: Sample data in JSON format

- **Data pre-processing**: For our model to understand the natural language input of Commonsense QA data set, we needed to tokenize the input data i.e. break the input data into discrete chunks of information . These chunks provided the vector representing the whole input data. The tokenization of each question and answer pair was performed using BERT Autotokenizer. After encoding the questions, the encoded questions were concatenated with the corresponding encoded answer. Additionally, each input in the data was padded with the max length of 128.

  To leverage the multi-processing capability of python, we divided our input data into mini-batches with each batch size of 4. The mini-batches were collated by stacking each input to the tensor with the help of a custom Collator function.
- **Fine-tuning of Pre-trained BERT Model for training**: A pre-trained BERT-base-uncased model having 12 layers with 110M parameters was used for the multiple choice classification task. Each layer of the model contains an encoder stack containing both encoder and decoder. This encoder-decoder network uses self-attention on the encoder side and attention on the decoder side. After applying self-attention, each layer passes the result through a feed-forward network before handing it to the next encoder. The intermediate units uses dropout of

0.1 and GELUActivation activation function. The final layer output is then passed through BERT pooler with Tanh activation. To bring linearity to the pooled output, a linear classifier is applied on the output which provides the requisite classification probabilities for each of the multiple choice answers. The loss function used is NLLLoss, which is a common loss function used in multi-classification problems. It takes the log of the probability value after activation function (softmax) and add the log probability value of the correct answer to the average.

The following training hyper-parameters were used during training:

- Learning rate: 5e-5
- Train batch size: 4
- Evaluation batch size: 4
- Optimizer: ADamW with weight decay of 0.01 applied to each of the bias parameters
- LR Scheduler Type: LambdaLR
- Number of epochs: 3
- Warmup steps: 500
- Output dimensions: 768

The learning rate was linearly increased from 0 to 5e-5 for the first 300 iterations and linearly decreased to 0 for the rest of the iterations. The model was trained on the train batch size of 4 and the evaluation was performed on the evaluation batch size of 4. A custom loss function was defined in the form of a Loss wrapper which uses NLLLoss. We also needed to update the model parameters while training based on the output of the loss function. To achieve this, we used ADamW optimizer . An optimizer bridges the gap between the loss function and model parameters by updating the model based on the output of loss function. We chose this optimizer because ADamW decouples the weight decay from the gradient update (Loshchilov and Hutter, 2017).

The linear increase and decrease of the learning rate was controlled by the learning rate scheduler. An LR scheduler adjusts the learning rate between epochs as the training progresses. We defined a custom function which linearly increased the learning rate for the first 300 iterations and then linearly decreased it. Since LambdaLR sets the learning rate of each parameter group to the initial learning rate multiplied by a given function, we used it as our LR scheduler.

The model utilized Logits for the classification purpose. Logits are the vector of un-normalized predictions generated by the model which were passed via softmax normalization function to generate the normalized probabilities with one value for each possible choice. The choice having the maximum probability was chosen as the predicted choice for calculating loss and accuracy.

## Evaluation Strategy

During the training and testing sets of the model we produce loss and accuracy evaluations. The test results, beginning with our baseline hyper-parameters, are listed below in figures 4 - 6:

```
flags.DEFINE_integer("train_batch_size", 32, "Total batch size for training.")

flags.DEFINE_integer("eval_batch_size", 8, "Total batch size for eval.")

flags.DEFINE_integer("predict_batch_size", 8, "Total batch size for predict.")

flags.DEFINE_float("learning_rate", 5e-5, "The initial learning rate for Adam.")

flags.DEFINE_float("num_train_epochs", 3.0,
                   "Total number of training epochs to perform.")
```

Figure 4: Baseline hyper-parameters

.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Accuracy | 21.46% | 20.56% | 21.87% | 20.96% | 19.90% | 19.57% |
| | | | | | | |
| | Baseline | 10 Epochs | 5.00E-04 | eval_batch | predict_bat | 5.00E-01 |
| | | | | | | |
| train_batch_size | 32 | 32 | 32 | 32 | 32 | 32 |
| eval_batch_size | 8 | 8 | 8 | 16 | 8 | 8 |
| predict_batch_size | 8 | 8 | 8 | 8 | 16 | 8 |
| learning_rate | 5.00E-05 | 5.00E-05 | 5.00E-04 | 5.00E-05 | 5.00E-05 | 5.00E-01 |
| num_train_epochs | 3 | 10 | 3 | 3 | 3 | 3 |

Figure 5: Fine-tuned Hyper-parameters Accuracy

.

```
I1207 01:50:23.377370 139831761168256 error_handling.py:115] evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
I1207 01:50:23.377567 139831761168256 run_commonsense_qa.py:757] ***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.21457821
I1207 01:50:23.377642 139831761168256 run_commonsense_qa.py:759]   eval_accuracy = 0.21457821
INFO:tensorflow:  eval_loss = 1.6090549
I1207 01:50:23.620012 139831761168256 run_commonsense_qa.py:759]   eval_loss = 1.6090549
INFO:tensorflow:  global_step = 913
I1207 01:50:23.620232 139831761168256 run_commonsense_qa.py:759]   global_step = 913
INFO:tensorflow:  loss = 1.6090775
I1207 01:50:23.620315 139831761168256 run_commonsense_qa.py:759]   loss = 1.6090775
```

Figure 6: Baseline results

.

Our goal was to fine tune the hyper-parameters to increase the accuracy of the model. But after testing several hyper-parameters we found any changes we made had a negligible result leading to the assumption that the hyper-parameters were already optimized, this stagnation can be seen in Figure 7. This assumption is supported by our failure to produce an update to the model architecture.

Without a way to improve our CommonsenseQA model or optimize its hyper-parameters we moved to looking for a solution with Deeplib. Deeplib was able to provide us with better results as seen in Figure 8 but we were not able to carry these improvements over to our other models. Deeplib has a higher accuracy of about 50 percent compared to our other model which was about 21 percent accurate.

## Conclusion

Text Classification and answer prediction is one of the prominent space where various machine learning algorithms are being applied. In our implementation, we fine-tuned pre-trained BERT base uncased model via Deeplib by linearly increasing the learning rate for a defined number of warmup steps and then linearly decreasing it for the rest of the iteration. This model used ADamW optimizer to update the parameters while training. The new fine-tuned model provided a significant increase from our baseline of 21% accuracy to 50% accuracy. Although we were not able to translate this improvement to the other models, this accuracy can

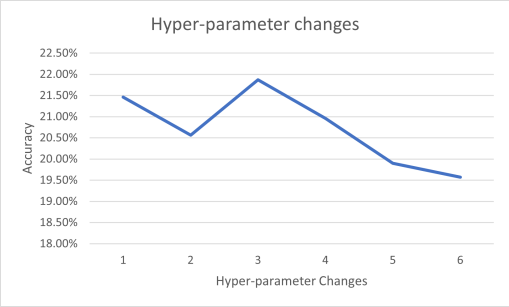


Figure 7: Model accuracy

.

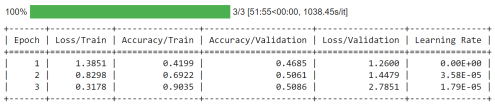


Figure 8: Deeplib accuracy

.

further be improved as part of the future implementation for this model. Future enhancements can be made by extending the proposed work with another new layer of CNN layer to potentially improve the accuracy. Apart from increasing the accuracy of BERT implementation, future works can also include implementation of other state-of-the-art techniques in machine learning (including but not limited to esim, alberta, and roberta).

## References

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

Floridi, L.; and Chiriatti, M. 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, 30(4): 681–694.

Hogan, A.; Blomqvist, E.; Cochez, M.; D'amato, C.; Melo, G. D.; Gutierrez, C.; Kirrane, S.; Gayo, J. E. L.; Navigli, R.; Neumaier, S.; Ngomo, A.-C. N.; Polleres, A.; Rashid, S. M.; Rula, A.; Schmelzeisen, L.; Sequeda, J.; Staab, S.; and Zimmermann, A. 2022. Knowledge Graphs. *ACM Computing Surveys*, 54(4): 1–37.

Ilya Loshchilov, F. H. 2017. Decoupled Weight Decay Regularization. In *"Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE); Optimization and Control (math.OC)"*. arXiv.

Lin, B. Y.; Chen, X.; Chen, J.; and Ren, X. 2019. KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning. arXiv:1909.02151.

Rajani, N. F.; McCann, B.; Xiong, C.; and Socher, R. 2019. Explain Yourself! Leveraging Language Models for Commonsense Reasoning. arXiv:1906.02361.

Si, C.; Molri, N.; Cheema, G.; Huang, E.; and Akkiraju, A. 2022. Benchmarking GPT-3 For Closed-Book QA: Strengths and Weaknesses.

Talmor, A.; Herzig, J.; Lourie, N.; and Berant, J. 2018. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. arXiv:1811.00937.

Xu, Y.; Zhu, C.; Xu, R.; Liu, Y.; Zeng, M.; and Huang, X. 2020. Fusing Context Into Knowledge Graph for Commonsense Question Answering. arXiv:2012.04808.

Yasunaga, M.; Bosselut, A.; Ren, H.; Zhang, X.; Manning, C. D.; Liang, P.; and Leskovec, J. 2022. Deep Bidirectional Language-Knowledge Graph Pretraining. arXiv:2210.09338.

Yichong Xu, S. W. S. S. H. C. X. L. J. G. P. H. M. Z. X. H., Chenguang Zhu. 2021. Human Parity on CommonsenseQA: Augmenting Self-Attention with External Attention. arXiv:2112.0325.

Zelikman, E.; Wu, Y.; Mu, J.; and Goodman, N. D. 2022. STaR: Bootstrapping Reasoning With Reasoning. arXiv:2203.14465.