EXTENDED ESSAY

Subject: Computer Science

Topic:

The Transformer architecture in common sense inference

Research Question:

To what extent is the state-of-the-art language model architecture Transformer accurate in commonsense inference tasks?

Word Count: 3941

May 2024

Table of Contents

1. Introduction

Natural language processing (NLP) is a subfield of Computer Science that applies Artificial Intelligence to understand and process human language. Priya et al. (2021) present a rich variety of applications of NLP such as sentiment analysis, healthcare, education, summarization, and translation, all of which aim to process text-based data in a more efficient and reliable manner than humans are capable of.

However, it is commonsense inference (CSI) that has gained a lot of attention recently in the field. Already in 1979, Schank showed that CSI is crucial for comprehending the world, enabling rational decision-making (Schank, 1975). Among other variations such as inferring the utterances in the dialogues (Ghosal et al., 2022) or assessing the truthiness of a statement (Onoe et al., 2021), CSI is primarily inferences based on commonsense knowledge about the physical worlds, that is, choosing the most likely real-world continuation of a particular context (Zhang et al., 2016). For instance, given the following sentence "Bob is repairing a car," we can infer that the continuation "he is taking off the tire" is much more plausible than "he is going for ice cream."

Over the past decade, considerable efforts have been made to construct datasets where the objective is to challenge modern language models on commonsense reasoning tasks. Compared to humans, these models performed poorly (Bowman et al., 2015; Wang et al., 2018) and it was not until the advent of Transformer architecture that the situation changed. Vaswani et al. (2017) proposed the state-of-the-art model architecture Transformer based solely on the self-attention mechanism which significantly outperformed previous models in many NLP tasks, including CSI.

Nevertheless, although powerful, the performance of the Transformer architecture in CSI tasks is still the subject of vigorous debate and uncertainty about the trust that humans can have in language models remains. Therefore, the following discussion shall analyze the research question: **To what extent is the state-of-the-art language model architecture Transformer accurate in commonsense inference tasks?**

2. Transformer architecture analysis

The Transformer architecture attains human-level performance in CSI through the self-attention mechanism, which effectively models long-range dependencies in text (Vaswani et al., 2017). In contrast, earlier architectures, such as Recurrent Neural Networks (RNN), face challenges in mapping dependencies between distant positions in a sequence due to their linear nature (Vaswani et al., 2017). RNNs calculate a sequence of hidden states h_t based on the previous hidden states h_{t-1} and the input at position t (Vaswani et al., 2017). As t increases, h_t contains diminishing context from the initial states/positions, posing a challenge for CSI (Vaswani et al., 2017). For instance, consider the example sentence "Bob is repairing a car." In CSI, knowing who is repairing the car is crucial but stated only at the beginning, creating an issue for RNNs.

Another advantage of the Transformer architecture is the incorporation of Position-wise Feed Forward Neural Networks (FFNs) (Vaswani et al., 2017). FFNs encode commonsense knowledge in their weights, which is vital for CSI (Colon-Hernandez et al., 2021).

Nonetheless, the way in which the elements of the original Transformer are assembled by Vaswani et al. (2017) is sub-optimal for CSI (Devlin et al., 2019), and a different structure is needed to achieve the best performance in CSI .

Therefore, this section will delve into explaining the self-attention mechanism and FFNs of the Transformer and their significance in CSI, as well as suggest and justify a different structure for the described elements.

2.1 Self-attention

Self-attention captures the relation between different positions in a given sequence, indicating how closely each word is related to others in a sentence—a significant step toward successful CSI. (Vaswani et al., 2017). Although the Transformer performs multiple attention functions rather than a single one, it is beneficial to firstly understand how a single attention head is calculated.

2.1.1 Single-Head attention

The self-attention function is called “Scaled Dot-Product Attention” (Fig.1). The input to this function is comprised of queries, keys, and values which are matrices $Q \in \mathbb{R}^{d_{seq} \times d_{model}}$, $K \in \mathbb{R}^{d_{seq} \times d_{model}}$, and $V \in \mathbb{R}^{d_{seq} \times d_{model}}$, respectively, where d_{seq} is the size of the input sequence.

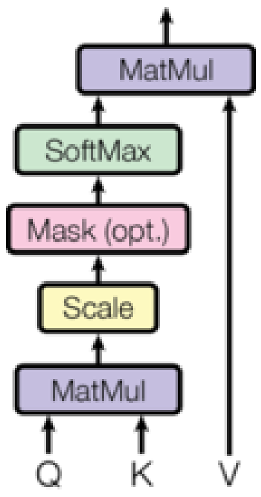
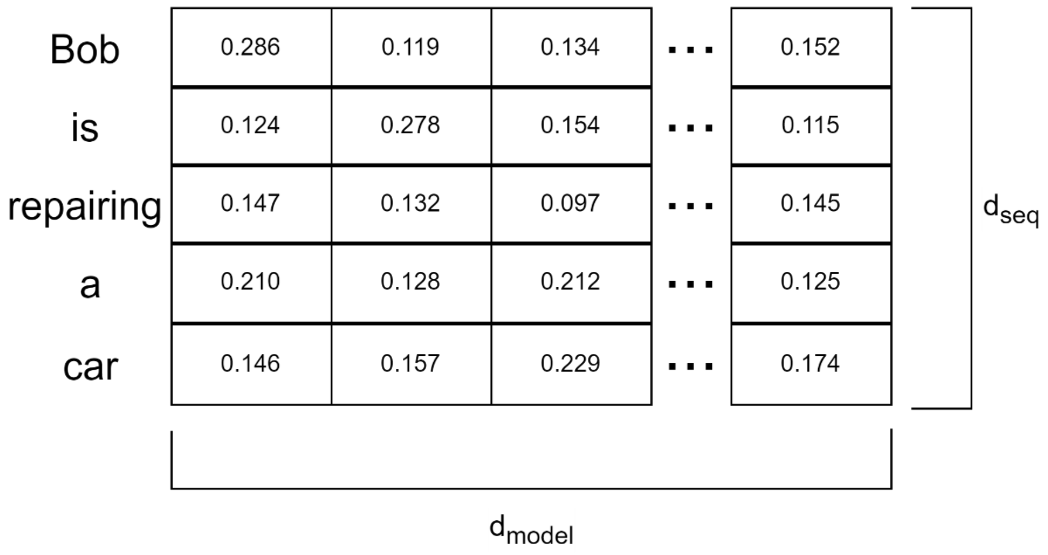


Figure 1. Scaled Dot-Product Attention. (adopted from Vaswani et al., 2017).

To calculate these matrices, embedding vectors are stacked on top of each other (Fig.2) to form a matrix X ∈ $\mathbb{R}^{d_{seq} \times d_{model}}$.



Bob	0.286	0.119	0.134	...	0.152
is	0.124	0.278	0.154	...	0.115
repairing	0.147	0.132	0.097	...	0.145
a	0.210	0.128	0.212	...	0.125
car	0.146	0.157	0.229	...	0.174

d_{seq}
d_{model}

Figure 2. Matrix of embeddings (numbers are random).

Then, three separate linear transformations are performed in which X is multiplied by three weight matrices $W^Q \in \mathbb{R}^{d_{seq} \times d_{model}}$, $W^K \in \mathbb{R}^{d_{seq} \times d_{model}}$, and $W^V \in \mathbb{R}^{d_{seq} \times d_{model}}$ such that:

$$Q = XW^Q \tag{1.1}$$

$$K = XW^K \tag{1.2}$$

$$V = XW^V \tag{1.2}$$

The weight matrices are learnable parameters[1].

The computation of the attention itself consists of the scalar product of Q and transpose of K (K^T), which is divided by the $\sqrt{d_{model}}$ factor (hence the dot product is called scaled). This scaled dot product is then the input to a $softmax$ function, which turn values in each word vector to values that sum to 1 (Wood, n.d.).

Lastly, the resulting matrix is multiplied by V. The process is given by the equation below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

[1] Different weights (or parameters, or bias) will be appearing throughout the discussion. They are all adjusted with backpropagation (Kurbiel, 2021). This mechanism is indeed interesting, but will not be examined as it goes beyond the scope of this investigation.

2.1.2 Multi-Head Attention

As previously noted, the Transformer comprises multiple attention layers, known as multi-head attention (MHA), facilitating a more detailed preservation of relations between distant words, a critical aspect in CSI (Vaswani et al., 2017). In MHA the queries, keys, and values are linearly projected h times (Fig.3) with various learned linear projections to d_k, d_k, and d_v, respectively, each of size d_{model}.

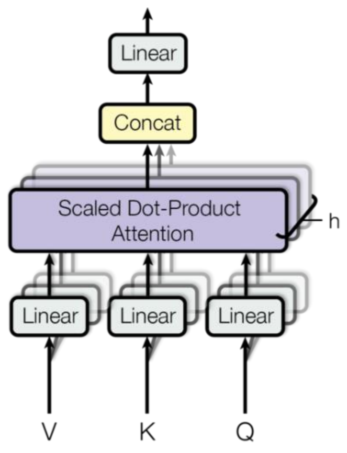


Figure 3. Multi-Head Attention (adopted from Vaswani et al., 2017).

The projection is done by multiplying Q, K, and V by parameter matrices and splitting the result into h vectors to which the attention function is applied separately and which are concatenated and multiplied by a matrix to achieve the initial dimensions:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \qquad (3.1)$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \qquad (3.2)$$

The parameter matrices are defined as follows: $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$.

2.2 Position-wise Feed-Forward Networks

The Position-Wise Feed-Forward Networks (FFNs) process the data by infusing commonsense knowledge through their weights, playing a crucial role in CSI (Phi, 2020). FFNs consist of two hidden layers (that contain weights) with a SeLU activation function in between (Vaswani et al., 2017) (Fig.4).

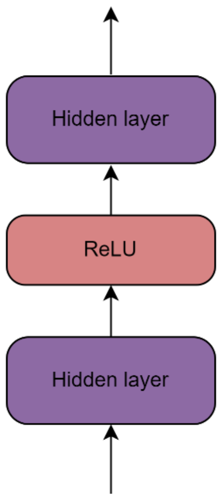


Figure 4. Position-Wise Feed-Forward Network.

The hidden layers are the intermediate representations of the input data and are matrices of size $\mathbb{R}^{d_{model} \times d_{ff}}$ and $\mathbb{R}^{d_{ff} \times d_{model}}$, respectively, where $d_{ff} = 4d_{model}$ (Vaswani et al., 2017). The calculation of a single hidden layer is done through matrix transformation and is given by:

$$T: \mathbb{R}^{d_{model} \times d_{ff}} \rightarrow \mathbb{R}^{d_{model} \times d_{ff}} \quad T(x) = W_q x + b_q \qquad (4)$$

where q is the layer number, W_q is the matrix of weights and b_q is bias (it shifts the function to better fit the data (Collis, 2017)).

These weights acquire commonsense knowledge through pre-training (Colon-Hernandez et al., 2021).

However, the two consecutive linear transformations (hidden layers) could be expressed as a single operation (Jung, 2023). Since CSI, and other tasks, require these transformations to be separate to inject more information into the data, an activation function that introduces non-linearity must be used (Jung, 2023). Unfortunately, some activation functions are prone to the vanishing gradient problem that limits how far a model can go with training, and thus, how much commonsense knowledge it can absorb (Krishnamurthy, 2022). To prevent the vanishing gradient problem, the Transformer architecture adopts ReLU as its activation function, which is also an optimal solution for CSI (Krishnamurthy, 2022).

The ReLU defaults all negative values to zero and leaves positive numbers as-is (Krishnamurthy, 2022). The ReLU function is given below:

$$f(x) = \max(0, x) \tag{5}$$

The overall computation of the FFN is therefore expressed by:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

where W_1 and b_1 are the parameters of the first hidden layer and W_2 and b_2 of the second.

2.3 Encoder-decoder structure limitations

The original implementation of the Transformer is intended for translation tasks and adopts the encoder-decoder structure (Vaswani et al., 2017) depicted in Fig.5.

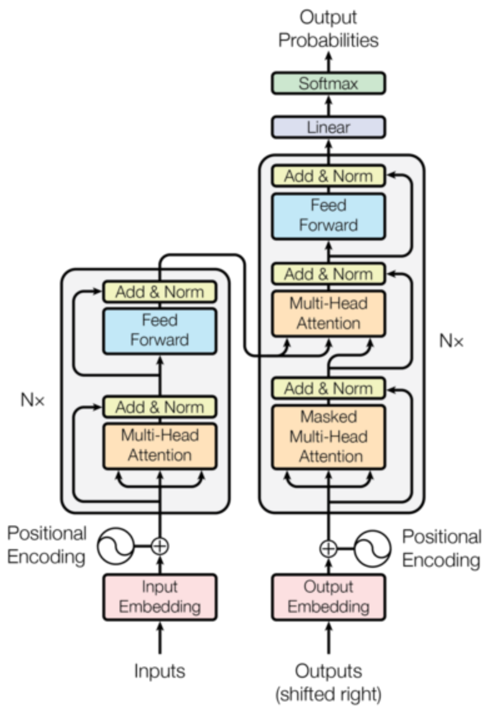


Figure 5. The Transformer architecture (adapted from Vaswani et al., 2017).

The encoder (left) and the decoder (right) share a similar structure, both containing the MHA and FFN sublayers (Vaswani et al., 2017). The decoder, however, introduces a third sublayer known as Masked Multi-Head Attention (MMHA) (Vaswani et al., 2017).

The distinction between MHA and MMHA lies in the preprocessing step. Before applying attention (3.1), MMHA replaces all illegal connections between the current word and the subsequent ones in the word matrix with $-\infty$ (mask). These values are later adjusted to 0 by the $softmax$ function (Fig.6).

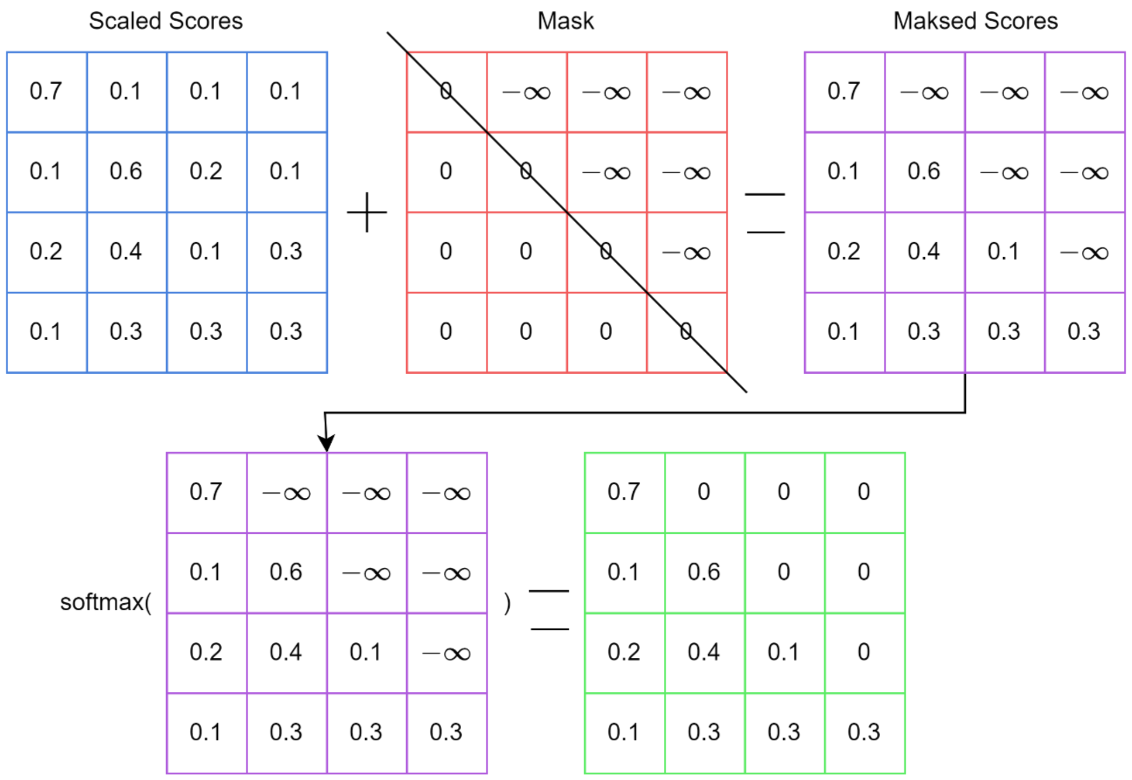


Figure 6. Simplified MMHA calculation.

This masking, or unidirectionality, found in the decoder is advantageous in translation tasks, where the architecture cannot condition on future tokens and must rely on context from only one side of the sentence (Vaswani et al., 2017). However, in sentence-level tasks such as CSI, bidirectionality is crucial as context from both sides of a sentence is needed (Devlin et al., 2019). For instance, given a pair of sentences: “The dog is chasing the cat” (A) and “He is having fun” (B), bidirectionality ensures that the model correctly associates the pronoun “he” in the sentence B with “the dog” in the sentence A, rather than “the cat.”

This necessitates the use of a Transformer-based model that inherits the basic units of the Transformer architecture, such as the MHA and FFN, but is bidirectional and adopts a different structure than the encoder-decoder for the purpose of CSI. In this regard, a sensible choice appears to be the Bidirectional Encoder Representation from Transformers (BERT) model, which is based solely on the Transformers' encoder and has gained immense popularity since its release by Google in 2019, particularly due to its integration into Google search (Nayak, 2019). Therefore, the following section will outline how BERT achieves bidirectionality and hence its importance in CSI.

3. BERT

BERT was pre-trained on BookCorps, a dataset composed of books (Zhu et al., 2015), and Wikipedia (excluding text passages with headers, lists, and tables) (Devlin et al., 2019). Nonetheless, the strength of BERT in CSI lies not only in the vast datasets used in pre-training but, more importantly, in the pre-training regimes (i.e., the Masked Language Model and Next Sentence Prediction) that were applied (Devlin et al., 2019). Additionally, the model is customizable toward a specific downstream task, in this case, CSI, through fine-tuning (Devlin et al., 2019). These three aspects will now be discussed.

3.1 Masked Language Model

Masked Language Model (MLM) is a pre-training task where some percentage of tokens is randomly masked and the goal is to predict what the masked tokens are (Taylor, 1953). In BERT, 15% of all tokens are masked, where 80% of these masked tokens are replaced with the [MASK] token, 10% are replaced with a random token, and 10% are left unchanged (Devlin et al., 2019). The above distribution is employed to mitigate the discrepancy between pre-training and fine-tuning that arises owing to the introduction of the additional token [MASK] in the former (Devlin et al., 2019).

Hence, to accurately predict the masked token, the model must comprehend the entire sentence context from both sides, as there are no hints about which tokens will be masked or if masking will occur. This way, MLM ensures bidirectionality and contributes to CSI.

3.2 Next Sentence Prediction

Next Sentence Prediction (NSP) is a pre-training task where the goal is to determine whether a sentence B is the continuation of a sentence A or not. In the context of BERT pre-training, B is the actual continuation of A (labelled as $IsNext$) 50% of the time, and, 50% of the time, it is not (labelled as $NotNext$) (Devlin et al., 2019).

NSP task trains the model to understand the relationship between sentences, a crucial aspect in CSI that is not achieved with other methods.

3.3 Fine-tuning

Although fine-tuning BERT is relatively inexpensive compared to pre-training, it has been demonstrated that fine-tuning is of utmost importance for downstream tasks such as CSI (Devlin et al., 2019). When fine-tuning for CSI, possible continuations are concatenated with [SEP] token and fed to as the input, and the correct answer is fed to as the output.

The next section will analyze the challenges of CSI datasets and choose an appropriate dataset for fine-tuning BERT.

4. Datasets

4.1 Challenges of CSI datasets

A common structure of CSI datasets consists of a premise p and three hypotheses $h_1,\ h_2,$ and h_3, and the goal is to determine whether h_i (i denotes the hypothesis) is semantically and logically entailed by p (Gururangan et al., 2018). The premise is typically derived from a video caption, which is a brief description of a real situation. The hypotheses, in three forms - entailment, neutral, and contradiction, are based on the provided premise. Entailment is definitely true given p, neutral may be true given p (but p does not provide such information), and contradiction is certainly not true given p (Gururangan et al., 2018).

However, research has shown that each type of hypothesis contains its unique annotation artifacts (stylistic patterns) that are introduced by humans. Language models can pick up on these artifacts and indicate the entailed hypothesis without looking at the premise (Gururangan et al., 2018). For instance, entailments usually have generic words such as “animal,” rather than specific such as “dogs,” while neutral hypotheses usually contain modifiers such as tall (Gururangan et al., 2018). Moreover, neutral hypotheses are, on average, the longest, while entailments are shorter than the neutral class, but longer than contradictions (Gururangan et al., 2018).

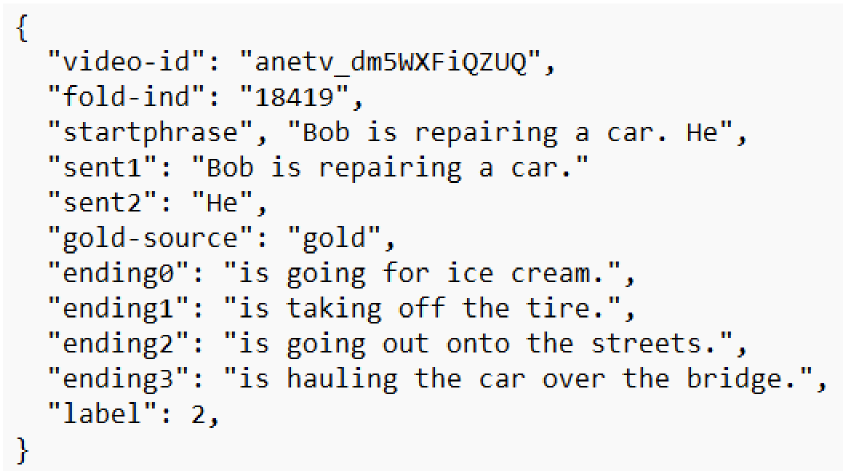
The presence of annotation artifacts raises the necessity for replacing exclusively human-generated content with less biased alternatives. The following section describes a possible solution.

4.2 SWAG

Zellers et al., (2018) have proposed SWAG (Situations With Adversarial Generations) dataset that introduces a novel approach for removing bias from CSI datasets - adversarial filtering. Adversarial filtering employs language models to generate endings (hypotheses) instead of humans (Zellers et al., 2018). However, before delving into the algorithm of adversarial filtering, the structure of SWAG must be explained.

4.2.1 The structure of SWAG

In SWAG, models are provided with a situation (premise) and their goal is to determine what is the next most likely event (hypothesis) to occur out of the four, which are given (Zellers et al., 2018). Mathematically, models are given some context $c = (s,\ n)$ (e.g. “Bob is repairing a car. He”), where s is a complete sentence (“Bob is repairing a car.”) and n is a noun phrase (“He”), and fours possible verb phrase sentence endings $V = \{v_1, \dots, v_4\}$ (such as “is taking off the tire.”) Figure 7, on the other hand, presents how one SWAG context (example) is stored in practice.



```
{
  "video-id": "anetv_dm5WXFiQZUQ",
  "fold-ind": "18419",
  "startphrase", "Bob is repairing a car. He",
  "sent1": "Bob is repairing a car."
  "sent2": "He",
  "gold-source": "gold",
  "ending0": "is going for ice cream.",
  "ending1": "is taking off the tire.",
  "ending2": "is going out onto the streets.",
  "ending3": "is hauling the car over the bridge.",
  "label": 2,
}
```

Figure 7. SWAG context storage.

The contexts and correct endings are drawn from two consecutive frames of a video captions that come from ActivityNet Caption and Large Scale Movie Description Challenge (Zellers et al., 2018). Although endings are not human-generated, they are human-verified to ensure that only one ending is plausible (Zellers et al., 2018).

Overall, SWAG consists of 113,000 such multiple choice questions (Zellers et al., 2018) where 73,000 is reserved for training, 20,000 for validation, and another 20,000 for test.

4.2.2 Adversarial filtering

As mentioned, in SWAG, humans do not generate candidate endings, but verify them. The task of constructing candidate endings is left to machine learning models, specifically a multilayer perceptron, bag-of-words, CNN, and Bidirectional Long Short-Term Memory Network (Zellers et al., 2018). However, since those models are trained on human-generated data (in this case on BookCorpus), they have annotation artifacts of its own (Zellers et al., 2018).

To counteract this, SWAG splits endings into verb phrase (VP) and noun phrase (NP) and uses the models to massively oversample candidate NPs that are stylistically similar to the original (Fig.8). Mathematically, there are N contexts, the original NP (positive example) is denoted as x_i^+ and candidate NPs (negative examples) are denoted as $x_{i,j}^-$ where $1 \leq j \leq N^-$ for each i (Fig.8).

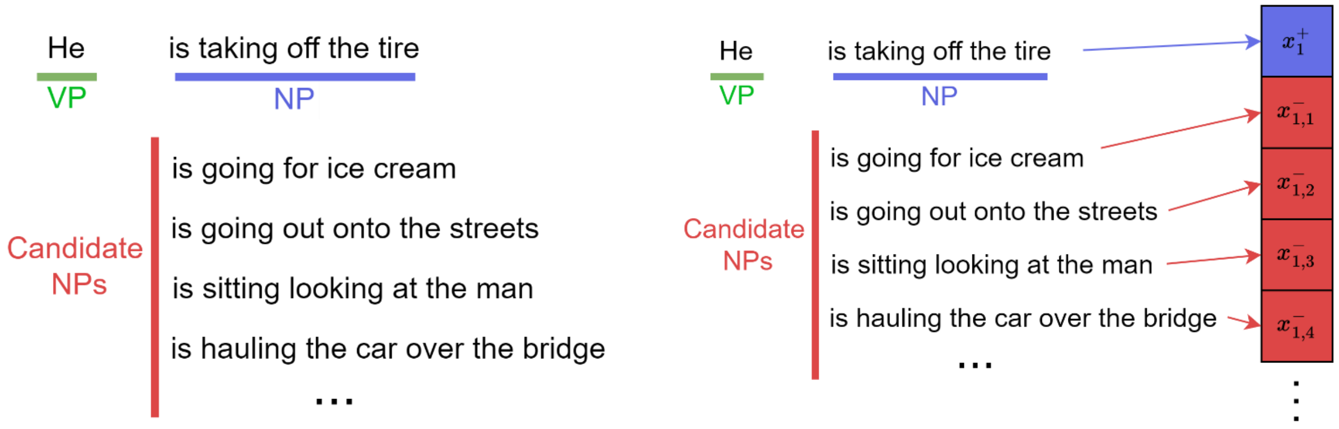


Figure 8. Oversampling candidate NPs.

Then, all N such vectors are joined together and divided into training and test sub-datasets. On the training sub-dataset, models learn which NPs were written by a human and which by a machine (the purpose is to remove annotations artifacts). After training, these models perform the same task on the test sub-dataset and replace machine-generated NPs with that written by humans. This process is repeated 100 times, altering training and test sub-datasets.

The formal definition of this process looks as follows. X is the input space (context), and Y is the label space (NPs). The trainable classifier is defined as $f_\theta : X \rightarrow \mathbb{R}^Y$ where θ is the parameters. The dataset is $D = \{(x_i, y_i)\}_{1 \le i \le N}$, and the lost function over the dataset is $L(f_\theta, D)$. Then, the set is said to be adversarial when the empirical error I is high:

$$I(D, f) = \frac{1}{N}\sum_{i=0}^{N} L(f_{\theta_i^*}, \{(x_i, y_i)\}), \qquad (7.1)$$

$$\text{where } \theta_i^* = argmin_\theta L(f_{\theta_i^*}, D \setminus \{(x_i, y_i)\}) \qquad (7.2)$$

This (7.1) should ensure that knowing $N - 1$ does not help to solve N.

Having described the model and dataset, the experiment to determine the accuracy of the Transformer architecture in CSI can be now conducted.

5. Experiment Methodology

5.1 Preprocessing

Recalling from the section 4.2.1 "The structure of SWAG", in SWAG, for each example, there is one context $c = (s,\ n)$ and four ending for each example $V = \{v_1, \dots, v_4\}$. However, since BERT accepts only the tokenized version of data, which is continuous, one must combine the above pieces together before tokenization via preprocessing.

The script for preprocessing SWAG is adopted from the Hugging Face documentation (Hugging Face, n.d.) and is written in Python (see 9.1 "Preprocessing script"). This functions performs three steps:

1. Creates a list of four copies of s (sent1, Fig.7) with n (sent2, Fig.7) attached to each of them for each example.
2. Creates a second list that consists of n joined with all four v_i (ending, Fig.7) for each example.
3. Flattens both lists to tokenize them and later unflattens them such that each examples has a corresponding $input\_id$, $attention\_mask$, and $labels$ field.

$input\_id$ is the example identifier, $attention\_mask$ identifies which part of a token is the actual sentence (the cause is the truncating in the tokenization phase,) and $labels$ indicates the correct answer.

5.2 Fine-tuning customization

Devlin et al. (2019) presented BERT in two versions: $BERT_{BASE}$ and $BERT_{LARGE}$, distinguished by their sizes. However, the experiment will employ the $BERT_{BASE}$, even though it is outperformed by $BERT_{LARGE}$ (Devlin et al., 2019), as $BERT_{LARGE}$ requires more than 16 GB of VRAM (GitHub, 2020) and the researcher had no financial resources to acquire such a machine.

$BERT_{BASE}$ has 12 layers ($N = 12$), 768 hidden units ($d_{model} = 768$), and 12 attention heads ($h = 12$).

Fine-tuning adjusts the biases and weights of the model and needs to be governed to ensure the model's performance. This is achieved through the use of a loss function that penalizes bad predictions: it is zero only when the prediction is perfect (in this case, the correct continuation is chosen), otherwise, it is greater (Google, n.d.). In other words, the goal is to find weights and biases such that the loss function is small. The experiment will use a cross-entropy function, widely used in multiclass classification tasks such as CSI (the four continuations) (Malnarowicz, 2021). Cross-entropy calculates the loss for each class label per observation and sums the results (Malnarowicz, 2021).

In addition, it is crucial to ensure the quality of the fine-tuning to prevent overfitting. Overfitting occurs when a model fits precisely against its training data, leading to underperformance against unseen data (IBM, n.d.). To mitigate overfitting, several measures can be taken, such as increasing data, adjusting the learning rate, and employing weight decay.

Data augmentation is one approach to artificially increase the dataset. An epoch, defined as one complete pass of the data forward and backward through the model (AI Wiki, 2020), can be utilized for this purpose. Conducting more than one epoch effectively increases the size of the dataset.

The learning rate governs the pace at which the values of parameter estimates are adjusted or learned (Google, 2022). A lower learning rate results in a more accurate but slower learning process.

Weight decay serves as a regularization technique by imposing a penalty on the weights to prevent them from becoming overly complex (Yang, 2020).

Furthermore, fine-tuning should be mindful of memory constraints. A batch size, representing the total number of training samples in a single batch, is used for this purpose (AI Wiki, 2020). A reasonable batch size ensures memory efficiency.

According to Devlin et al. (2019) and Hugging Face documentation, optimal values for these parameters, avoiding overfitting and enhancing memory efficiency, are 3 epochs, a batch size of 16, a learning rate of 5e-5, and weight decay of 0.01.

The dataset is split into training, test, and validation datasets according to its default configuration (see 4.2.1 "The structure of SWAG").

The script for fine-tuning is written in Python (see 9.2 "Fine-tuning").

6. Experimental results

6.1 Accuracy across epochs

Accuracy is calculated on the test dataset after each epoch as follows: Accuracy = (TP + TN) / (TP + TN + FP + FN), where: TP: True positive TN: True negative FP: False positive FN: False negative.

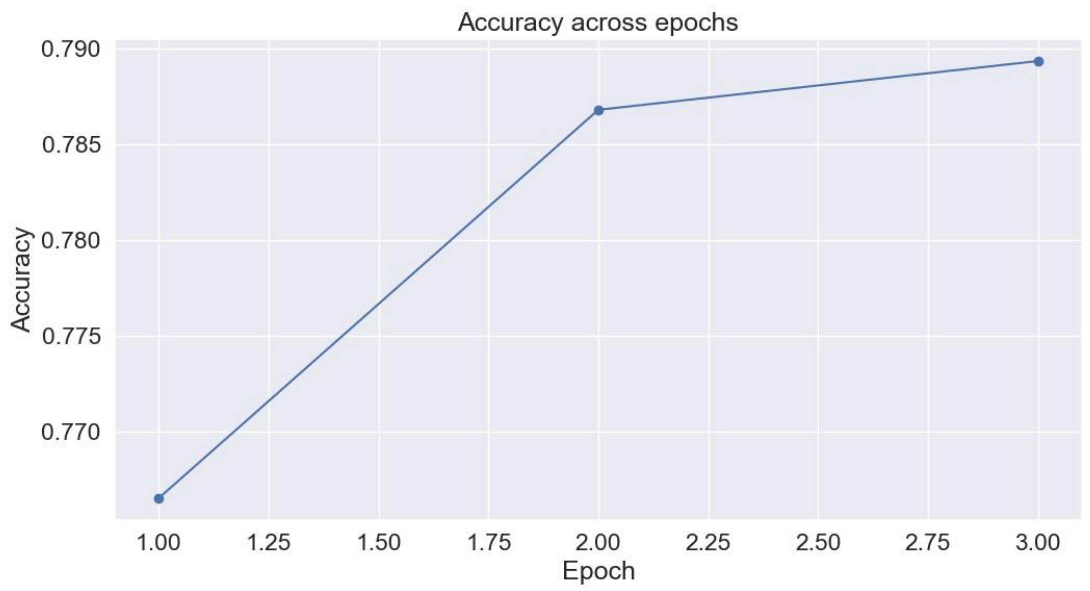


Figure 9. Accuracy across epochs.

6.2. Training loss across epochs

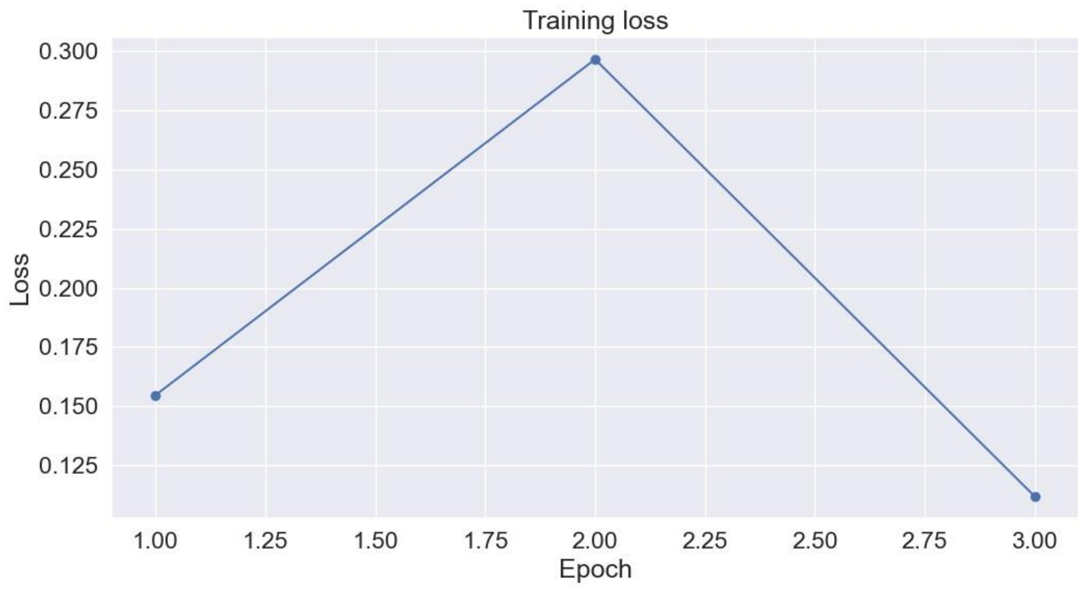


Figure 10. Training loss across epochs.

7. Analysis & Conclusion

The model has achieved 79% accuracy (Fig.9) with minimal loss (<0.125) (Fig.10) in the last epoch. This result could be even improved by employing $BERT_{LARGE}$ (Devlin et al., 2019) with more epochs (Zellers et al., 2019). However, achieving a 79% accuracy, while not quite matching the human performance of 88% (Zellers et al., 2018), suggests that the state-of-the-art language model architecture Transformer, solely based on the self-attention mechanism, performs CSI tasks with a very high degree of accuracy.

This is particularly promising, especially with the emergence of numerous BERT-based models (Lan et al., 2020; Liu et al., 2019). These models incorporate various enhancements, such as modifications to the MLM pre-training objective, resulting in superior performance compared to BERT in many tasks. They hold the potential to elevate the performance of the Transformers architecture in CSI to unprecedented levels (Lan et al., 2020; Liu et al., 2019). Consequently, further research is essential to explore the capabilities of these models in CSI, evaluating the accuracy of the Transformer architecture and determining whether it can achieve performance levels indistinguishable from human capabilities.

Nevertheless, certain limitations of this research concerning CSI need to be discussed. Despite the efforts in this investigation to reduce distributional information that contaminates CSI datasets by selecting SWAG with its AF to assess the model's commonsense abilities rather than detecting language patterns, it has been uncovered that some biases may still endure in SWAG (Zellers et al., 2019). This is evident from the fact that the accuracy of BERT on SWAG, with various parameters, decreases only slightly when the model is not provided with context (Zellers et al., 2019). This sheds light on the fact that, although the Transformer architecture displays a considerable level of accuracy in CSI, it actually hallucinates knowledge since it has no real understanding of the problem.

This becomes even more apparent when applying BERT that was fine-tuned on SWAG, assessing the model's understanding of the physical world, to another dataset evaluating a different type of commonsense knowledge, such as CICERO, which tests models on their abilities to infer utterances in a dialogue. In spite of high accuracy on SWAG, BERT fine-tuned in this way achieves low accuracy on CICERO (Ghosal et al., 2022). This further emphasizes the aforementioned hallucination with the Transformer architecture: it gives the impression of understanding by attaining great accuracy in the niche CSI task on which it was fine-tuned. However, in reality, the Transformer architecture does not possess commonsense knowledge and has limited applicability in CSI, as its use cannot be extended beyond its training dataset.

Overall, the Transformer architecture offers clear advantages over other model types. In addition to structural benefits, such as self-attention, its training is standardized, enabling global experimentation with novel modeling approaches. Indeed, innovations in this field emerge regularly, enhancing the architecture's performance, including CSI. However, the effectiveness of these models in CSI relies on the specific dataset used, limiting the widespread commercial use and stunting their development in this area. The implementation of commonsense knowledge, as presented in this paper, is unlikely to experience rapid further development. Notably, major AI advancements often originate from large companies, rather than research teams from universities. A case in point is the fact that Google introduced BERT, and other models are merely improvements on the core architecture.

8. Bibliography

AI Wiki. (2020). *Epochs, Batch Size, & Iterations*.

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). *A large annotated corpus for learning natural language inference*. Association for Computational Linguistics. http://aclweb.org/aclwiki/index.php?

Collis, J. (2017, April 14). *Glossary of Deep Learning: Bias*. Deeper Learning.

Colon-Hernandez, P., Havasi, C., Alonso, J. B., Huggins, M., & Breazeal, C. (2021). Combining pre-trained language models and structured knowledge. *CoRR*, *abs/2101.12294*. https://arxiv.org/abs/2101.12294

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://github.com/tensorflow/tensor2tensor

Ghosal, D., Shen, S., Majumder, N., Mihalcea, R., & Poria, S. (2022). *CICERO: A Dataset for Contextualized Commonsense Inference in Dialogues*.

GitHub. (2020, June 16). *How much GPU memory needed for the Bert-large model?*

Google. (n.d.). *Descending into ML: Training and Loss*.

Google. (2022). *Reducing Loss: Learning Rate*.

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. R., & Smith, N. A. (2018). *Annotation Artifacts in Natural Language Inference Data*.

Hugging Face. (n.d.). *Multiple choice*.

IBM. (n.d.). *What is overfitting?*

Jung, T. (2023). The Linear and Nonlinear Nature of Feedforward. *The Feynman Journal*.

Krishnamurthy, B. (2022, October 28). *An Introduction to the ReLU Activation Function*. Builtin.

Kurbiel, T. (2021). Spatial Transformer Networks — Backpropagation. *Towards Data Science*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*.

Malnarowicz, D. (2021). *Loss Functions*.

Nayak, P. (2019, October 25). *Understanding searches better than ever before*. Google.

Onoe, Y., Zhang, M. J. Q., Choi, E., & Durrett, G. (2021). *CREAK: A Dataset for Commonsense Reasoning over Entity Knowledge*.

Phi, M. (2020, April 28). *Illustrated Guide to Transformers Neural Network: A step by step explanation*.

Priya, B., Nandhini, J. M., & Gnanasekaran, T. (2021). An analysis of the applications of natural language processing in various sectors. *Advances in Parallel Computing*, *38*, 598–602. https://doi.org/10.3233/APC210109

Schank, R. C. (1975). Using Knowledge to Understand. In B. L. Nash-Webber & R. Schank (Eds.), *Theoretical Issues in Natural Language Processing*. https://aclanthology.org/T75-2023

Taylor, W. L. (1953). "Cloze Procedure": A New Tool for Measuring Readability. *Journalism Quarterly*, *30*(4), 415–433. https://doi.org/10.1177/107769905303000401

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*. http://arxiv.org/abs/1706.03762

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. http://arxiv.org/abs/1804.07461

Wood, T. (n.d.). *Softmax Function*. DeepAI.

Yang, S. (2020, October 4). *Deep learning basics — weight decay*. Analytics Vidhya.

Zellers, R., Bisk, Y., Schwartz, R., & Choi, Y. (2018). *SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference*.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). *HellaSwag: Can a Machine Really Finish Your Sentence?*

Zhang, S., Rudinger, R., Duh, K., & Van Durme, B. (2016). *Ordinal Commonsense Inference*. http://arxiv.org/abs/1611.00601

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. *2015 IEEE International Conference on Computer Vision (ICCV)*, 19–27. https://doi.org/10.1109/ICCV.2015.11

9. Appendix

9.1 Preprocessing script

(adopted from (Hugging Face, n.d.))

```
from huggingface_hub import login
from datasets import load_dataset
from transformers import AutoTokenizer


login(token="token_name")


model_checkpoint = "bert-large-uncased"
batch_size = 16
datasets = load_dataset("swag", "regular")
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint, use_fast=True)


ending_names = ["ending0", "ending1", "ending2", "ending3"]



def preprocess_function(examples):
    # Repeat each first sentence four times to go with the four possibilities of second
    sentences.
    first_sentences = [[context] * 4 for context in examples["sent1"]]
    # Grab all second sentences possible for each context.
    question_headers = examples["sent2"]
    second_sentences = [[f"{header} {examples[end][i]}" for end in ending_names]
    for i, header in enumerate(question_headers)]

    # Flatten everything
    first_sentences = sum(first_sentences, [])
    second_sentences = sum(second_sentences, [])

    # Tokenize
    tokenized_examples = tokenizer(first_sentences, second_sentences,
    truncation=True)
    # Un-flatten
    return {k: [v[i:i + 4] for i in range(0, len(v), 4)] for k, v in
    tokenized_examples.items()}


encoded_datasets = datasets.map(preprocess_function, batched=True)
```

9.2 Fine-tuning script

(adopted from (Hugging Face, n.d.))

```
from transformers import AutoModelForMultipleChoice, TrainingArguments,
Trainer

from dataclasses import dataclass

from transformers.tokenization_utils_base import PreTrainedTokenizerBase,
PaddingStrategy

from typing import Optional, Union

import torch

import numpy as np
```

```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
model = AutoModelForMultipleChoice.from_pretrained(model_checkpoint)
model.to(device)

model_name = model_checkpoint.split("/")[-1]
args = TrainingArguments(
    f"{model_name}-finetuned-swag",
    evaluation_strategy = "epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    weight_decay=0.01,
    push_to_hub=True,
)


@dataclass
class DataCollatorForMultipleChoice:
    """
    Data collator that will dynamically pad the inputs for multiple choice received.
    """
```

```
tokenizer: PreTrainedTokenizerBase
padding: Union[bool, str, PaddingStrategy] = True
max_length: Optional[int] = None
pad_to_multiple_of: Optional[int] = None

def __call__(self, features):
    label_name = "label" if "label" in features[0].keys() else "labels"
    labels = [feature.pop(label_name) for feature in features]
    batch_size = len(features)
    num_choices = len(features[0]["input_ids"])
    flattened_features = [[{k: v[i] for k, v in feature.items()} for i in
    range(num_choices)] for feature in features]
    flattened_features = sum(flattened_features, [])

    batch = self.tokenizer.pad(
        flattened_features,
        padding=self.padding,
        max_length=self.max_length,
        pad_to_multiple_of=self.pad_to_multiple_of,
        return_tensors="pt",
    )

    # Un-flatten
    batch = {k: v.view(batch_size, num_choices, -1) for k, v in batch.items()}
    # Add back labels
```

```
        batch["labels"] = torch.tensor(labels, dtype=torch.int64)
        return batch



accepted_keys = ["input_ids", "attention_mask", "label"]
features = [{k: v for k, v in encoded_datasets["train"][i].items() if k in
accepted_keys} for i in range(10)]
batch = DataCollatorForMultipleChoice(tokenizer)(features)



def compute_metrics(eval_predictions):
    predictions, label_ids = eval_predictions
    preds = np.argmax(predictions, axis=1)
    return {"accuracy": (preds == label_ids).astype(np.float32).mean().item()}


trainer = Trainer(
    model,
    args,
    train_dataset=encoded_datasets["train"],
    eval_dataset=encoded_datasets["validation"],
    tokenizer=tokenizer,
    data_collator=DataCollatorForMultipleChoice(tokenizer),
    compute_metrics=compute_metrics,
)


trainer.train()
```