BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Knowledge enhanced language models

vorgelegt von

Alexei Gustavo Figueroa Rosero

EDV.Nr.:877737

dem Fachbereich VI
der Beuth Hochschule für Technik Berlin vorgelegte Masterarbeit
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)
im Studiengang
Data Science

Tag der Abgabe 27. Juli 2020

Gutachter

Prof. Dr.-Ing. habil. Alexander Löser	Beuth Hochschule für Technik Berlin
Prof. Dr. Felix Alexander Gers	Beuth Hochschule für Technik Berlin

Abstract

This work presents a methodology to enhance state of the art transformer models with additional knowledge, by introducing an intermediate *retraining* routine in a multi-task setup. Every task is derived from external knowledge base triples. The knowledge domain chosen is commonsense due to its relevance, data and benchmark availability. The experimental results highlight dropout as an important setting to control the retraining process, and show promising avenues to overcome the selected benchmark.

Contents

Chapter 1

Introduction

The field of Natural Language Processing (NLP) has sustained an increasingly rapid transformation over the last couple of years. Not only has it captured the undeniable full attention of industry tech giants such as Google, Facebook, Amazon and Microsoft, which indeed are also contributors of the general advancement, but also has settled as a main focus point for research given the wide spectrum of its applicability spanning many domains outside of IT services. Just as in all the related branches of Machine Learning, the developments in Neural Network architectures, hardware and software frameworks have become a fundamental factor in the fast iterations pushing forward the state of the art.

The most recent wave of progress driving the best NLP systems has been mainly characterized by very deep Neural Network architectures, an ever exploding number of parameters growing in parallel with larger and larger datasets. This trend has somewhat limited the exploration of such models to the tech industry leaders that have the viability in the sense of both economic and technical capacity. Nevertheless, the lessons learned from Transfer Learning have transcended NLP, and in conjunction with the publication of the parameters of such large general purpose models after training, have lead the research community to advance the field into specializing further on a wide range of different language tasks. This separation of the final NLP problems into a broad and very expensive *pretraining* of language models, and a shorter, but highly specialized *finetuning*, poses the question of where is the right boundary between these two stages specially regarding what data and in what amount should be used for each one of them.

Deep Neural Networks generally exhibit a lack of comprehensive interpretability and the best pretrained NLP models are no exception to this. Dissecting where and how the knowledge of the world and its semantics are stored in the parameter space of a large model is still only an unmaterialized dream, so adjusting methodologically the knowledge contents at either *pretraining* or *finetuning* of these models, is a process that functionally involves model fitting and an assessment against a commonly accepted benchmark for a specific NLP task.

This work presents an intermediate training regime that would instill additional knowledge into a pretrained model relevant to a domain specific finetuning task. This is motivated by the fact that highly specialized domain data is not necessarily available in an amount that is relevant for it to be simply part of the *pretraining* process, and this is also forgiving the fact that *pretraining* at a large scale is economically not feasible for most of the research community. A more in-depth discussion of the scope of this work is presented next.

1.1 Problem statement

Recent results in Adversarial Datasets have shown that the State Of The Art (SOTA) models in NLP have not been an exception of the *clever Hans* phenomenon, learning unintended statistical features of the data used to train them, instead of generalizing to use a principled approach to solve problems. Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2018] made the headlines when it achieved near-human performance on the commonsense benchmark SWAG [Zellers et al., 2018], nevertheless, by later improvement of these adversarial datasets, BERT was shown to have exploited *annotation artifacts*. In NLP commonsense remains an important dimension of knowledge since it is so tightly connected to how humans solve ambiguities in their use of language. Further improvements in teaching this knowledge to language models is highly desirable and many benchmarks as well as specialized knowledge bases are available.

This work combines the ingredients of SOTA transformer models, knowledge bases, and a multitask setup to introduce a retraining regime that lies in between the processes of *pretraining* and *finetuning* to instill additional knowledge into these models. The commonsense knowledge is targeted because of the availability of knowledge bases and downstream tasks, but the proposed methodology is intended to be compatible with any kind of domain knowledge. Formally, the experiments of this work are designed to verify the next hypothesis.

1.1.1 Hypothesis

Given a pretrained model, such as, BERT-base-uncased, and a retraining with multiple tasks derived from commonsense knowledge bases, the retrained model will outperform a model of the same architecture without retraining when put through a commonsense downstream task. Additionally, this retraining procedure will not affect significantly the capability of the model at general language tasks.

1.2 Thesis outline

This thesis is composed of the following 5 chapters:

- **Background and Related work**: Presents the background and parallel research relevant to the thought process behind the hypothesis and the experimental setup for its testing.
- **Methods**: Illustrates the design of the experiments ran and the justification behind the decisions made.
- **Implementation**: Summarizes the technical details of the experimental setup and illustrates the full experiment flow.
- **Results**: Highlights the quantitative results of the experiments and compares them to the chosen baseline, additionally presents an error analysis of the outcome of the experiment.
- **Summary and future work**: Depicts the extent and limitations of this work and poses open questions to further explore in the context of the proposed hypothesis.

Chapter 2

Background and Related Work

In order to lay the foundations for the methods and experiment design of this work, few important related notions need to be put in context. This chapter will first explore the recent transition of the state of the art of NLP from recurrent encoder-decoder architectures to transformers, then elaborate on a few of the problems in the commonsense dimension that this work attempts to solve with the use of specific commonsense knowledge bases and downstream tasks.

2.1 Recurrent networks

Recurrent Neural Networks (RNN) involve a feedback of the hidden layers, in the sense of forward computation, to the inputs. Such networks are highly relevant at modeling sequential data, since a stateful representation of the process being modeled is stored in the hidden layers and is constantly used to compute an output.

A breakthrough in the architecture of such networks was to include gating mechanisms to control in a differentiable manner the flow of the input, output and state information. An example is the Long Short Term Memory LSTM network which includes a definition of input, output, state and forget gates [Hochreiter and Schmidhuber, 1997, Gers et al., 1999]. This network has seen wide usage in the modeling of sound, time-series, and in general: sequences. The intuition behind the design of these networks is to mitigate the problems of gradient explosion and vanishing which made the training of recurrent networks fairly difficult, as well as to improve the modelling of long term dependencies in the sequences [Hochreiter et al., 2001].

Although RNNs are theoretically very capable architectures (Turing-complete [Siegelmann and Sontag, 1992]), and they can learn from and map to arbitrarily long sequences, they are not well suited for parallelization, thus their training can take a significant amount of time.

2.1.1 RNNs in Language Models

The Neural Machine Translation field introduced the use of LSTMs in an encoder-decoder pipeline. Here, the encoder would incrementally create a latent representation of the input sequence that would be decoded and finally reinterpreted as words. Figure 2.1 presents this architecture as originally proposed in [Cho et al., 2014].

The sequence-to-sequence learning tasks of finding latent representations or *embeddings* of natural language sequences are now commonly referred to as *language modelling* [Graves, 2013, Sutskever et al., 2011, Sutskever et al., 2014, Cho et al., 2014].

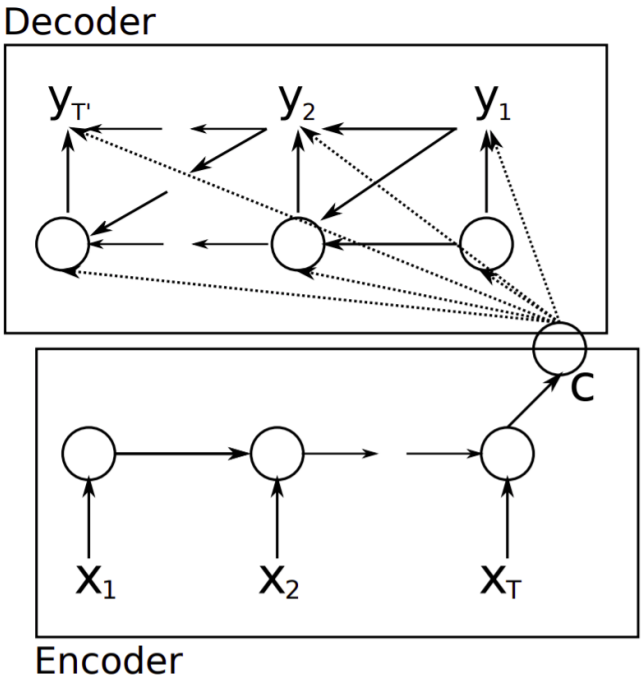

Figure 2.1: Encoder-decoder architecture for RNNs from [Cho et al., 2014].

2.2 Transformer networks

Transformer networks were introduced as a step towards tackling the issues of time complexity at the moment of training recurrent networks, they are parallelizable feed-forward networks that rely on *attention* mechanisms and don't require the expensive sequential inference used with recurrent models.

Attention was originally introduced in recurrent models for Neural Machine Translation so that the models would be better at mapping the relationships between input and target sequences by *attending* selectively to the source inputs [Bahdanau et al., 2014].

[Vaswani et al., 2017] proposed the Transformer architecture that, while it reuses the spirit of the sequence to sequence modelling of having an encoder and a decoder, discards completely the notion of recurrence by using encoder-decoder attention and *self-attention*, . Figure 2.2 shows the original Transformer. The publication of this model not only meant a significant improvement in the State Of The Art SOTA metrics in Neural Machine Translations tasks, but also, a more scalable and faster architecture, superseding the RNN based models.

2.3 State Of The Art

As of 2020 the SOTA in NLP is dominated by Transformer-based networks, they are increasingly large in their number of parameters to an extent that has made them exclusive to a select number of players with the economical means to maintain the proportionally large technical requirements. The recently introduced GPT-3 made it to the headlines with an outstanding 175 Billion parameters, but what is still an important aspect is that the performance of such models still scales without reaching the point of diminishing returns [Brown et al., 2020].

An important aspect of the later models has been their *pre-training* and *finetuning* regimes. The

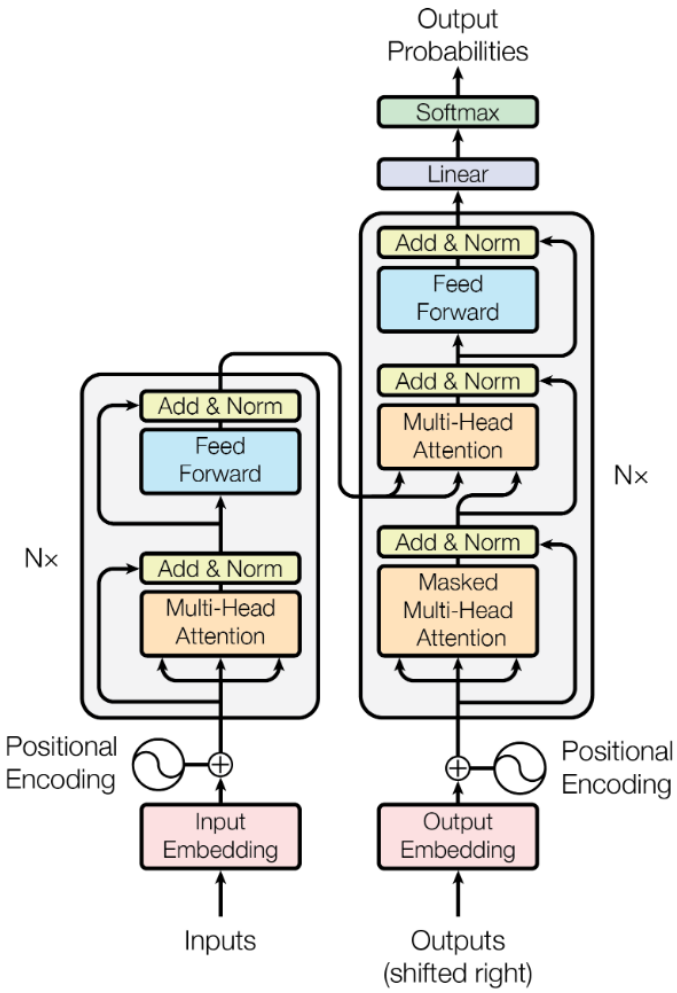

Figure 2.2: Transformer architecture from [Vaswani et al., 2017].

pre-training stage is mostly unsupervised and involves tasks like next word prediction, next sentence prediction, and predicting masked words within sentences. This is done on very large text corpora expecting for the model to achieve a high degree of knowledge of the language. *Finetuning* is subjecting the pretrained model to an actual relevant task with domain specific data like sentiment analysis or question answering among others. Usually *finetuning* is done on considerably smaller datasets and over a few epochs. This separation of these two stages has made possible for these large and thoroughly pretrained models to be used and researched in many fields by players that not necessarily have the massive hardware or data to run the *pre-training*, yet they can focus on more specialized tasks.

2.3.1 BERT

Bidirectional Encoder Representation from Transformers (BERT) is a mutilayer transformer model that is based on the work of [Vaswani et al., 2017]. The main architectural difference in contrast to other comparatively large transformer-based models is the bidirectional self-attention, i.e. attention that is not constrained to the left of the context of every token [Devlin et al., 2018]. Empowered with this and the two *pretraining* tasks of next-sentence-prediction and masked-language-modelling, BERT set the bar across all NLP benchmarks when published.

Figure 2.3 illustrates how the *pre-training* and *finetuning* of BERT take place. In the sense of

architecture there is no actual difference between these two settings besides the output layers or *heads*. This characteristic became a crucial step into generating a working framework within the research community to transfer the language knowledge learned during *pre-training* to an ever growing set of application fields or downstream tasks.

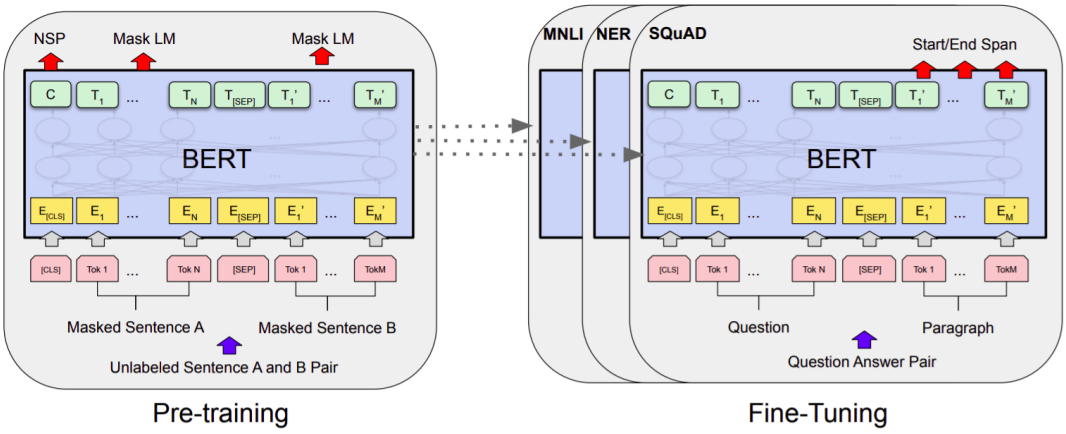

Figure 2.3: BERT, pre-training and finetuning settings differ only in the configuration of the output layers, `[CLS]` is a special token denoting the start of a sequence while `[SEP]` delimits two sequences, from [Devlin et al., 2018].

As mentioned before the pre-training of BERT is composed of two different tasks:

- **Masked language modelling MLM** where a proportion of the source tokens chosen at random is masked with the special token `[MASK]`, and the objective for the network is to predict them.

- **Next sentence prediction**, given two sentences separated by the `[SEP]` token the objective is for the network to predict whether the second sentence follows the first one, this is approached as a binary classification problem.

2.4 Catastrophic forgetting

The subdivision of the training of a Deep Neural Network (DNN) model into multiple consecutive stages (e.g. *pretraining* and *finetuning*) might yield severe degradation of performance in one of these tasks. This is very relevant to this work since the methodology involves an additional training step.

Catastrophic forgetting is a problem of neural networks when trained consecutively on two or more different tasks. It occurs when the network loses its capability to perform on the initial task once it has been retrained. Such pattern can be explained by the fact that the parameters learned for the first task are replaced at the moment of specializing the network on the next ones [McCloskey and Cohen, 1989].

Strategies of regularization like dropout and choosing the nature of the tasks might hinder the extent of catastrophic forgetting [Goodfellow et al., 2013].

2.5 Commonsense

Humans ground information of the world constantly through the diverse input from their senses, and while context is fairly relevant for disambiguation on a case to case basis in the sense of language understanding, there are cues that, although regularly used, have been built over the conscious lifetime of an individual and are not necessarily present in the window of language context. This implicit or background knowledge is also referred to as commonsense knowledge.

Although Deep Learning techniques have in recent years addressed to a *successful* extent problems in the visual, speech and NLP domains, arguably many of the SOTA models fail when needed to incorporate commonsense knowledge. In contrast, humans are very adept at generalizing on few learning examples and render trivial questions such as *If you stick a pin into a carrot, does it make a hole in the carrot or in the pin?* [Davis and Marcus, 2015], where many neural networks would find it hard to decide between the *carrot* or the *pin*.

[Marcus, 2018] in his critical appraisal of Deep Learning (DL) postulates a set of important aspects regarding models, data and knowledge.

- DL is data hungry.
- DL is shallow and has limited capacity of transfer.
- DL has no natural way of dealing with hierarchical structure.
- DL struggles with open-ended inference.
- DL is not sufficiently transparent.
- DL has not been well integrated with prior knowledge.
- DL cannot inherently distinguish causation from correlation.
- DL presumes a large stable world, in ways that may be problematic.
- DL works well as an approximation, but its answers cannot be fully trusted.
- DL is difficult to engineer with.

Many of these problems are subject of active research on their own, yet all seem very fundamental if commonsense were to be infused in a model.

2.6 HellaSwag

HellaSwag is a novel benchmark for commonsense Natural Language Inference (NLI) that uses Adversarial Filtering AF, a technique in which a set of discriminators are trained to choose from wrongly generated samples from a language model to iteratively create a multiple choice question answering dataset that is almost obvious for humans and very challenging for machine models [Zellers et al., 2019b].

AF was originally the subject of SWAG [Zellers et al., 2018], the preceding attempt of creating a dataset that would remove *annotation artifacts* [Gururangan et al., 2018], i.e. stylistic patterns of the data such as length and word-preference biases that might leak the information of labels to models under evaluation purely with features of the dataset.

With the introduction of BERT, SWAG was rendered trivial, reaching even human performance levels on the benchmark. However, HellaSwag showed that SWAG was overcome by BERT not because of BERT's capacity in terms of NLI, but rather because of the cues already present in the data. To demnostrate this, as shown in figure 2.4, different versions of SWAG were created:

- **Default**: The default SWAG dataset.
- **Ending Only**: All the context was removed from the questions.
- **Shuffled**: The tokens per ending in a question are shuffled.
- **Shuffled + Ending only**: Context is removed and additionally the order of the tokens is randomized for each ending.

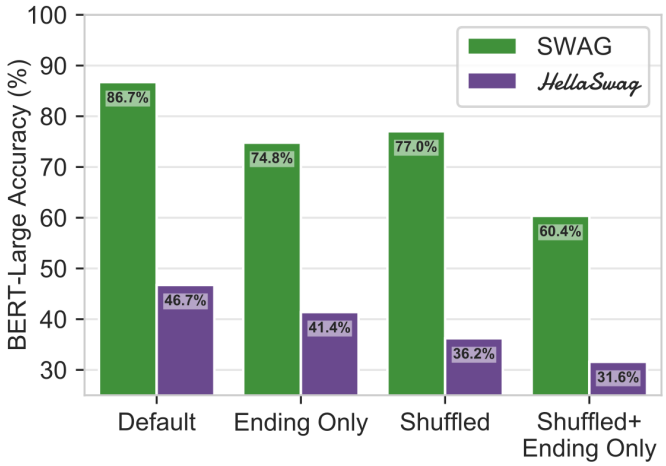

Figure 2.4: BERT validation accuracy when trained under several versions of SWAG and HellaSwag. BERT's performance on SWAG changes slightly, even with very aggressive alterations in the question structure, hinting to the presence of learned statistical features of the data instead of actual commonsense reasoning. [Zellers et al., 2019b]

Even with the most aggressive alteration of the datasets (Shuffled + Ending only) BERT managed to surpass ESIM+ELMo [Chen et al., 2016] and LSTM+ELMo [Zellers et al., 2018] : the former SOTA when SWAG was published.

The contribution of HellaSwag lies mainly on the substitution of both the generators and discriminators with more recent SOTA models such as OpenAI's GPT and BERT. When evaluated on this novel benchmark, BERT struggled to breach 50% in overall accuracy. A single sample from HellaSwag is shown in Figure 2.5.

2.7 GLUE

GLUE is a Natural Language Understanding NLU benchmark that emphasizes the use of a diverse set of tasks to evaluate the performance of models [Wang et al., 2018b]. It is a collection of 9 different tasks and corresponding datasets. These tasks can be categorized as: single-sentence, similarity and paraphrase, and inference:

2.7.1 Single-Sentence

- **CoLA** Corpus of Linguistic Acceptability, each example of which is a sentence labelled with its grammatical correctness [Warstadt et al., 2019].

Category: Sharpening knives (ActivityNet; Zero-Shot)

Two men are in a room and the man with a blue shirt takes out a bench stone and with a little lubricant on the stone takes an knife and explains how to sharpen it. then he

a) uses a sharpener to smooth out the stone using the knife. (100.0%)
b) shows how to cut the bottom with the knife and place a tube on the inner and corner. (0.0%)
c) bends down and grabs the knife and remove the appliance. (0.0%)
d) stops sharpening the knife and takes out some pieces of paper to show how sharp the knife is as he cuts slivers of paper with the knife. (0.0%)

Figure 2.5: Context and endings from HellaSwag with their probabilities as evaluated with BERT, in red is BERT's answer and in bold the golden label. [Zellers et al., 2019b]

- **SST-2** Stanford Sentiment Treebank, a sentiment, *positive/negative*, is predicted from movie reviews.

2.7.2 Similarity and paraphrase

- **MRPC** Microsoft Research Paraphrase Corpus, automatically retrieved from online news sources. Consists of sentence pairs and the task is to predict whether they are semantically equivalent.
- **QQP** Quora Question Pairs, involves a collection of question pairs and the goal is to determine if they are semantically equivalent.
- **STS-B** Semantic Textual Similarity Benchmark is a collection of sentence pairs retrieved from news headlines, video and image captions. The task is to assess the sentence similarity.

2.7.3 Inference

- **MNLI** Multi-Genre Natural Language Inference Corpus is a crowdsourced selection of sentence pairs with textual entailment annotations. Given premises and hypothesis sentences the task is to predict whether the premise entails, contradicts or is indifferent to the hypothesis.
- **QNLI** The Stanford Question Answering Dataset consists of question-paragraph pairs, where one of the sentences in the paragraph contains the answer to the corresponding question. QNLI is a modified dataset that maps each sentence of the paragraph to a pair of context and question sentences, yielding a sentence-classification problem where the goal is to predict if the answer to the question is contained in the context [Wang et al., 2018b].
- **RTE** Recognizing Textual Entailment, comes from a compilation of annual textual entailment challenges. The task involves detecting entailment in two classes *entailment* and *no_entailment*.
- **WNLI** The Winograd Schema Challenge. Given a sentence containing a pronoun a system must choose from a list the referent of that pronoun. For WNLI, the ambiguous pronoun is replaced with every choice, and a sentence pair of the original sentence and the one with the replacement is constructed. The task is to predict if the sentence with the replacement entails the original sentence.

Figure 2.6 lists all the GLUE tasks, data size and their corresponding evaluation metrics.

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Figure 2.6: Task descriptions and metrics used for GLUE [Wang et al., 2018b]

2.8 Knowledge Bases

In the context of NLP knowledge can be either extracted from corpora or it can be manually compiled and curated. One of the greatest challenges of automatic knowledge extraction from corpora is *reporting bias*, i.e. the frequency of claims within these corpora are not necessarily sensible to truth or facts of real life [Gordon and Van Durme, 2013]. On the other hand manually curated knowledge bases typically present high correspondence or precision towards facts, but lack coverage. A common technique to tackle this is by performing Knowledge Base Completion where suitable novel relations are created among the entities of the knowledge base.

In this work two knowledge bases are used, ATOMIC and CONCEPTNET, both of them correspond to curated graphs, yet they differ in the level of abstraction of the entities and relations they contain.

2.8.1 ATOMIC

ATOMIC is an atlas of everyday commonsense reasoning, composed of 877k textual descriptions of knowledge. The main contribution of ATOMIC is that it is a curated knowledge graph that does not namely express taxonomic knowledge, but rather a set of *if-then* like relations focused on causality of events, agents, themes and mental states [Sap et al., 2019].

There are 3 groups of relations in this graph that correspond to 9 edges:

- **If-Event-Then-Persona**: This group of relations describe how the subject is perceived (*xAttr*).
- **If-Event-Then-Event**: These relations encompass events that might follow other events (*xNeed, xWant, oWant, xEffect, oEffect*).
- **If-Event-Then-MentalState**: These edges relate to the mental pre and post-conditions of an event (*xIntent, xReact, oReact*).

Figure 2.7 lists a few examples of the relations of ATOMIC based on this categorization.

An additional way of classifying the types of relations of ATOMIC is by their causal relations i.e. causes, effects and stative (attributes). Figure 2.8 illustrates a small sub-graph from ATOMIC with these in mind.

Event	Type of relations	Inference examples	Inference dim.
"PersonX pays PersonY a compliment"	If-Event-Then-Mental-State	PersonX wanted to be nice	xIntent
		PersonX will feel good	xReact
		PersonY will feel flattered	oReact
	If-Event-Then-Event	PersonX will want to chat with PersonY	xWant
		PersonY will smile	oEffect
		PersonY will compliment PersonX back	oWant
	If-Event-Then-Persona	PersonX is flattering	xAttr
		PersonX is caring	xAttr
"PersonX makes PersonY's coffee"	If-Event-Then-Mental-State	PersonX wanted to be helpful	xIntent
		PersonY will be appreciative	oReact
		PersonY will be grateful	oReact
	If-Event-Then-Event	PersonX needs to put the coffee in the filter	xNeed
		PersonX gets thanked	xEffect
		PersonX adds cream and sugar	xWant
	If-Event-Then-Persona	PersonX is helpful	xAttr
		PersonX is deferential	xAttr
"PersonX calls the police"	If-Event-Then-Mental-State	PersonX wants to report a crime	xIntent
		Others feel worried	oReact
	If-Event-Then-Event	PersonX needs to dial 911	xNeed
		PersonX wants to explain everything to the police	xWant
		PersonX starts to panic	xEffect
		Others want to dispatch some officers	oWant
	If-Event-Then-Persona	PersonX is lawful	xAttr
		PersonX is responsible	xAttr

Figure 2.7: Examples of **If-Event-Then-X** commonsense knowledge present int ATOMIC. For Inference dimensions, "x" and "o" pertain to PersonX and others, respectively (e.g. "xAttr": attribute of PersonX, "oEffect": effect on others) [Sap et al., 2019]

2.8.2 CONCEPTNET

ConceptNet is a knowledge graph where its nodes or *terms* are words and phrases of natural language which are connected by weighted edges or *assertions*. It was originally released in 2004 as a parsed representation of the crowd-sourced Open Mind Common Sense knowledge project.

ConceptNet has been enhanced to include common sense of many sources, domains and languages cite [Speer et al., 2017]. In total ConceptNet contains over 21 million edges and 8 million nodes, the English vocabulary alone is represented by 1.5 million nodes. ConceptNet 5.5 groups relations into 36 core edges, both *symmetric* and *asymmetric* (directed):

- **Symmetric**: *Antonym, DistinctFrom, EtymologicallyRelatedTo, LocatedNear, RelatedTo, SimilarTo, Synonym*
- **Asymmetric**: *AtLocation, CapableOf, Causes, CausesDesire, CreatedBy, DefinedAs, DerivedFrom, Desires, Entails, ExternalURL, FormOf, HasA, HasContext, HasFirstSubevent, HasLastSubevent, HasPrerequisite, HasProperty, InstanceOf, IsA, MadeOf, MannerOf, MotivatedByGoal, ObstructedBy, PartOf, ReceivesAction, SenseOf, SymbolOf, andUsedFor*

Figure 2.9 presents 4 different relations and their terminal nodes for the word *bicycle* in the 5.8th version of ConceptNet.

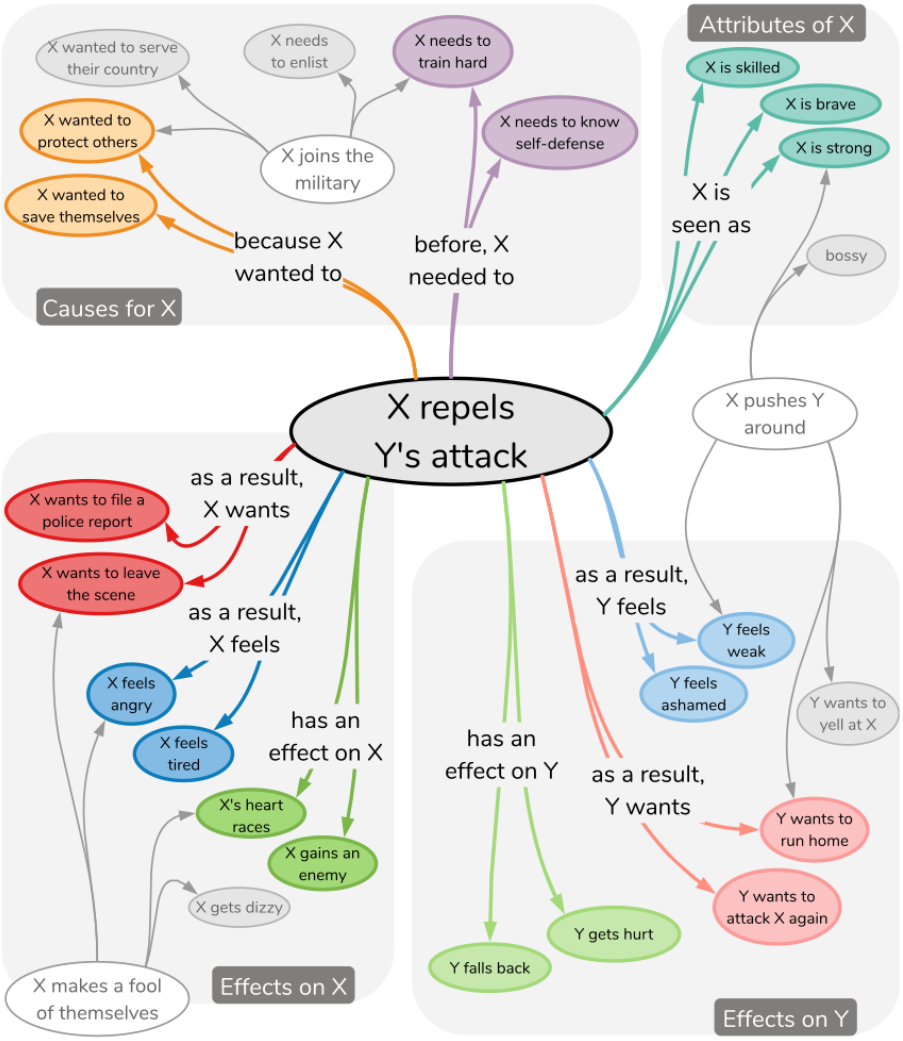

Figure 2.8: An example of a subset of triplets of ATOMIC, with the grey overlay the edges are grouped depending on their causal nature, i.e. effects, causes and attributes [Sap et al., 2019].

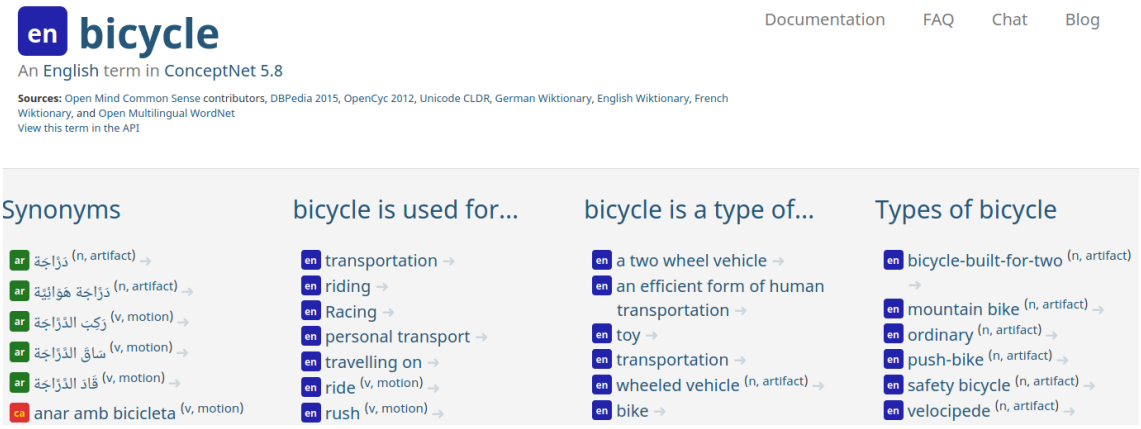

Figure 2.9: Sample of 4 different relations and their terminal nodes linked to the english word bicycle in ConceptNet5 [ConceptNet5, 2020].

2.9 Summary

This chapter introduces the building blocks for the experiments conducted in this work. Transformer networks and namely BERT are briefly presented as the SOTA in NLP related tasks. The notion of commonsense is explored by its definition, as well as, by artificially created datasets like HellaSwag that challenge the state of the SOTA and expose BERT as a model that exploits *annotation artifacts*. Additionally, two common-sense knowledge graphs are introduced: ATOMIC, composed with highly curated triples consisting of if-then like relations and CONCEPTNET, with a much broader coverage of common-sense knowledge. Finally, GLUE as an important general purpose language understanding benchmark is introduced.

Chapter 3

Methods

The design choices of this work encompass three core topics:

- A retraining routine
- A multi-task model with the corresponding task definitions
- An evaluation task

These will be discussed next.

3.1 Knowledge instillation through retraining

[Peters et al., 2019] propose a methodology for augmenting BERT with knowledge bases using an additional architectural component called Knowledge Attention and Recontextualization KAR. This module is inserted in between a pair of hidden layers of a pre-trained BERT. In spirit KAR capitalizes on an entity linker that retrieves precomputed embeddings from the knowledge base, then via word-to-entity-span attention the BERT embeddings are enriched.

A similar approach is taken by [Wang et al., 2020] with a module called K-Adapter, that might be seen as an additional parameter set that is finetuned with a specific knowledge base and downstream task, while selectively keeping parts of the original BERT frozen.

Both of these approaches encompass the modification and namely extension of the architecture of the pre-trained models. This involves an increased amount of parameters, higher complexity, training overhead and finally, slower inference times of the resulting models.

Recent results in model distillation like DistillBERT [Sanh et al., 2019], and research towards questioning the necessity of all the parameters in the SOTA transformer models (e.g. excessive number of heads [Michel et al., 2019]), have highlighted the redundancy existing in models like BERT. Appending additional components like K-Adapters or KAR pushes even further the complexity of such models, and the fact that both are intrinsically using transformer based sub networks poses the question of whether BERT could simply re purpose its redundancy to capture information from a domain specific Knowledge Base.

BERT is originally pre-trained in a multitask setting with two tasks: next sentence prediction and masked language modelling [Devlin et al., 2018]. The finetuning generally involves an additional set of output layers that implement directly to the downstream task.

[Garg et al., 2019] and [Phang et al., 2018] have shown applications of intermediate retraining (or finetuning) that improve the performance of BERT in the final downstream task. [Wang et al., 2018a] presents a comprehensive study on the intermediate retraining of BERT and concludes that:

- Generally a multi-task setting is shown to be more beneficial than any single task.
- Target downstream tasks do not necessarily have to be similar to the retraining tasks to benefit from them.
- When trained on tasks that are very different from its pre-training tasks (like Machine Translation MT) BERT shows signs of catastrophic forgetting.

Inspired by these conclusions this work aims to implement an intermediate retraining regime that involves a multitask setting and inherently follows the spirit of the two pre-training tasks of BERT. This retraining regime aims to instill specific domain knowledge with the help of tasks that generate or extend a knowledge graph.

As highlighted in Chapter 2, commonsense is an important type of knowledge that SOTA models including BERT are missing. With HellaSwag [Zellers et al., 2019b], proposes not only a critical diagnostic of BERT but also a downstream task that can be used to benchmark the model's proficiency in relation to commonsense knowledge. This is the main reason why HellaSwag is chosen to evaluate if the retraining instillation of commonsense knowledge bases is successful.

The choice of the commonsense knowledge bases in this work (ATOMIC and CONCEPTNET) resonates around the idea of having a fairly abstract and inference-focused base (ATOMIC), as well as very rich and general purpose knowledge (CONCEPTNET). Combining them falls in line with having a heterogeneous and large amount of data that should be in service of both the generalization and regularization of the model.

The details surrounding the design of the wrapping multitask model for the retraining and the selection of tasks are presented next. Even though the choice of pretrained model for this work lies with BERT it is important to denote the retraining regime proposed with these tasks can also be applicable to other SOTA models. The same logic applies to different domain knowledge, i.e. knowledge bases coming from specific domains like medical or legal, could also be instilled, to be later benchmarked against a corresponding domain specific downstream task.

3.2 Multitask Model

The model to be retrained is BERT-base and the methodology of the intermediate retraining regime follows the conventional downstream task finetuning of [Devlin et al., 2018]. However, the multitask setting implies that multiple objectives are optimized simultaneously during retraining. This is achieved by connecting to the last hidden state of BERT a set of additional layers, *output heads*, that output predictions in parallel. Every training sample that is fed into the model corresponds uniquely to a single task, but since all the output heads are concurrently yielding predictions for all samples in a batch, during training the losses are masked with the exception of the relevant task, so that the corresponding gradients can be propagated selectively. Figure 3.1 illustrates this retraining process for all the tasks that will be introduced next.

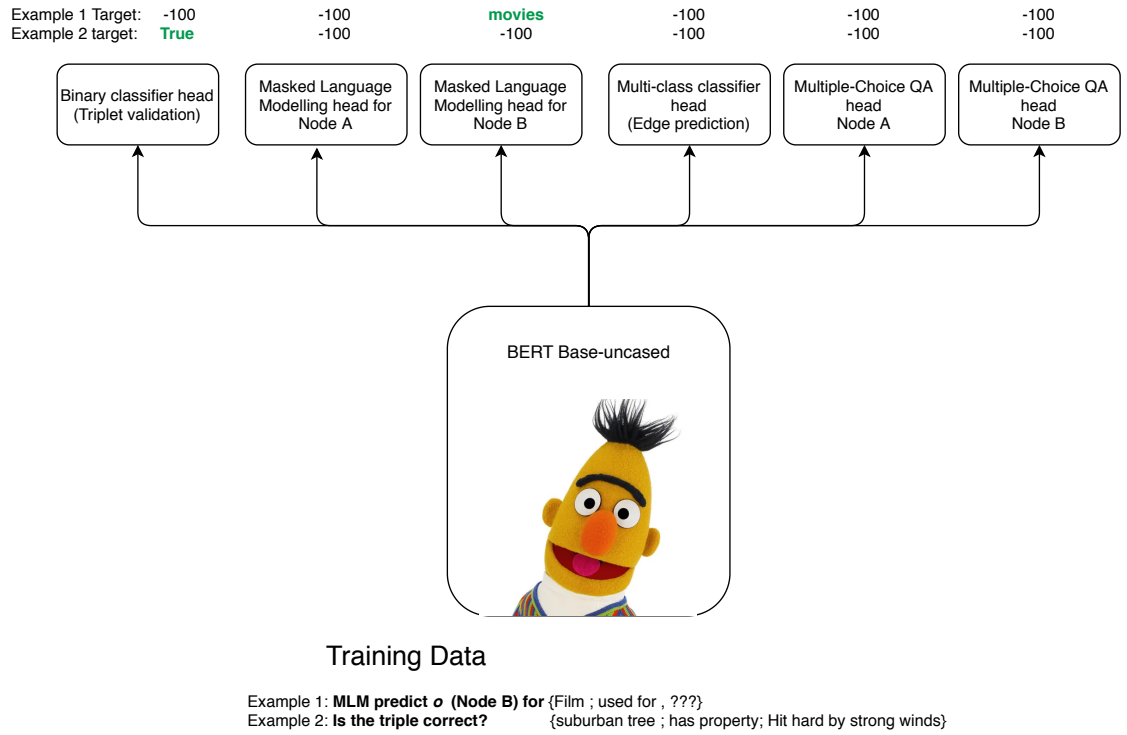

Figure 3.1: Training setup of the model with the six different output heads. The training examples are fed for a single task and the output targets are masked accordingly so as to propagate the gradient of the loss only to the corresponding output head of the model as well as to BERT Base-uncased

.

3.2.1 Knowledge Graph Representation

In order to set a common notation for this chapter the instillation knowledge graphs are represented as a set of triples $\{s, r, o\}$ s and o are graph nodes and r denotes the relation or edge between them. Figure3.2 shows a small sample graph that follows this notation. Both ATOMIC and CONCEPTNET are treated following this definition.

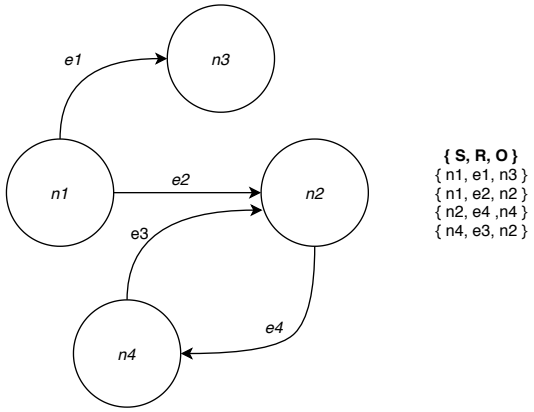

Figure 3.2: $\{s, r, o\}$ triples in a sample directed graph, the nodes are qualified as s and o depending on the direction of the edge or relation r

.

3.3 Tasks

The intuition behind the choice of tasks follows Knowledge Base Completion routines used to pretrain embeddings:

- In linear models [Joulin et al., 2017] (r edge prediction and node s, o prediction).
- The original transformer model [Bosselut et al., 2019](right node o prediction).
- In LSTM based networks [Li et al., 2016] (triplet validation).

These tasks show a degree of similarity to the two original pre-training tasks of BERT [Devlin et al., 2018], this idea is explained within the definition of each task and shows the motivation behind the choice of tasks to be consistent with the findings of [Wang et al., 2018a].

All the tasks presented next involve at least one additional output head fed with the last hidden layer of the model.

3.3.1 Triplet validation

This task involves the prediction of whether an s, r, o triple given to the encoder is valid or not. Given arbitrary triples s, r, o the model must predict whether they belong to any of the triples contained in the knowledge graph.

At an architecture level this is achieved by adding a softmax layer mapping to two classes, and consuming the output of the last hidden layer of the model. The model prediction function is defined in Equation 3.1.

$$f(s, r, o) \mapsto \mathbb{R}^2_{(0,1)} \tag{3.1}$$

During training, the loss L optimized for this task is the cross entropy between the prediction of the model and a one hot encoded vector of size 2 symbolizing the expected classes *correct* and *not correct* (or present and not present in the knowledge bases).

$$L = -y \cdot log(\hat{y}) \tag{3.2}$$

where y is the ground truth and $\hat{y}$ the model prediction.

Given the fact that the sample triple s, r, o is represented as three consecutive sequences, this task deeply resembles the original pre-training task of next sentence prediction of [Devlin et al., 2018].

3.3.2 Edge prediction

Similarly, the edge prediction involves a classification task. It consists of a softmax layer following the last hidden layer of the encoder to represent the probabilities over the classes. However, in this case the relation r is the actual target. Hence, the prediction function for this task only depends on s and o as shown in Equation 3.3.

$$f(s, o) \mapsto \mathbb{R}^C_{(0,1)} \tag{3.3}$$

where C is the number of relations or edges existing in the data. The loss of the model during training in this case remains the same as in Equation 3.2, albeit with the corresponding predictions and targets relevant to this task.

In the edge prediction task, BERT is presented with a single sequence composed of s and o, for the knowledge bases considered in this work the cardinality of r is significantly smaller than the set of all s and o, or even BERT's pretraining vocabulary, so even when it would be possible, it does not seem reasonable to treat this task as Masked Language Modelling MLM. Nevertheless, this task could indeed be thought of a simplified version of MLM.

3.3.3 Node prediction via Masked Language Modelling

For this task either the s or o nodes of the triple are masked out and the model is given the task of completing them. The prediction of the network involves a distribution over the *vocabulary size* of the network for each of the possible positions in the output sequence. Since this task is subdivided into two sub tasks, (for each of the nodes), there are two output heads. Equations 3.4 and 3.5 present the prediction of these heads:

$$f(r, o) \mapsto \mathbb{R}_{(0,1)}^{S \times V} \tag{3.4}$$

$$f(r, s) \mapsto \mathbb{R}_{(0,1)}^{S \times V} \tag{3.5}$$

where S is the size of the output sequence and V is the size of the vocabulary of the model.

The way these output heads are implemented is by mapping to a distribution (in the softmax sense) over the vocabulary for each of the output tokens. The loss at training time is the cross-entropy loss from Equation 3.2.

This task mimics very closely the MLM task in [Devlin et al., 2018].

3.3.4 Multiple choice

The multiple choice setting corresponds as well to a classification problem, yet again the task is subdivided into two sub tasks, one for each of the nodes of the triplet s or o. The input for the model is a sequence composed of:

- $s, r, c_1, c_2, ..., c_n$ where c_i corresponds to $n-1$ randomly sampled o nodes from the training dataset and the original o that belongs with the triple, n is the hyperparameter representing the number of choices.
- $o, r, c_1, c_2, ..., c_n$ where c_i and n follow the aforementioned meaning but for the s entities within the training data.

The main assumption made here is that the directional nature of the relations r is not relevant to the inference of the correct choice by the model. Additionally, each of the two output heads is indeed an n classification problem that will be treated as the tasks of edge prediction and triplet validation presented earlier, i.e. the output is produced by a softmax layer mapping to a distribution over n classes, and the loss is the cross-entropy loss of Equation 3.2.
This task can be thought of as a simplification of the previously presented masked language modelling retraining task, since the choices are now limited to n instead of a sequence of combinations over the entire BERT vocabulary.

3.4 Model evaluation

As described before the knowledge instillation is expected to be achieved after an intermediate multitask retraining routine. Subsequently whether the knowledge has been infused is assessed by running a commonsense specific benchmark - HellaSwag. Additionally to assess the state of the model after retraining the GLUE benchmark is also considered.

3.4.1 Retraining

In the multitask setting, each of the 6 different output heads corresponding to the 4 tasks described, have different target metrics:

Task	Task name	Evaluation metric
Triplet validation	IS_CORRECT	F1-Score
Edge prediction	MASK_EDGE	F1-Score
Masked language modelling of s	MASK_A	Perplexity
Masked language modelling of o	MASK_B	Perplexity
Multiple choice of s	MC_NODE_A	F1-Score
Multiple choice of o	MC_NODE_B	F1-Score

Table 3.1: Tasks of the retraining, their naming and the metrics used to control them

- For the classification tasks the main metric observed is the F1 score
- For the two node prediction tasks the control metric is the perplexity.

As stated before these metrics are not the main goal of this work, they only present a means to assess whether the retraining converges positively for each of the tasks, hence the eventual meeting of good metrics in this regard doesn't imply any success or failure in the subsequent evaluation with the downstream tasks. Table 3.1 presents a summary of the tasks and their metrics, additionally each of the 6 tasks is named.

Additionally, the model is put through two evaluation benchmarks corresponding to common sense and general language proficiency.

3.4.2 Common sense : HellaSwag

As previously introduced HellaSwag is a multiple-choice task, samples of which involve a context with 4 different endings. Figure 2.5 illustrates an example of such tasks. The general metric for HellaSwag as proposed by [Zellers et al., 2019b] is accuracy.

3.4.3 General language proficiency : GLUE

The GLUE benchmark is commonly used as a reference for assessing the language understanding of SOTA models. Given that both [Phang et al., 2018] and [Wang et al., 2018a] use GLUE in their findings, it is a benchmark of interest to assess the retrained model. As stated in Chapter 2 GLUE is a benchmark of 9 different datasets and tasks. Figure 2.6 presents more details on the evaluation metrics behind each of these tasks.

3.5 Summary

This chapter presented the design choices made to retrain and further fine-tune the model following the ideas of [Phang et al., 2018], [Garg et al., 2019] and [Wang et al., 2018a].
Four different tasks of graph-triplet validation, edge or relation prediction, node prediction and a simplification of node prediction via multiple choice question answering are described. In terms of architecture of the model these tasks involve six output heads connected to the last layer of the model, four of which correspond to a softmax classification and two to masked language modelling.

The initial retraining metrics target the two types of heads, being F1-score for classification and perplexity for the language modelling task. These metrics are only a means to control the success of the retraining phase.

The end goal of the methods presented is to evaluate if the model after retraining has improved its knowledge in relation to two knowledge bases. These knowledge bases are ATOMIC and CONCEPTNET which contain different perspectives of commonsense knowledge.

To evaluate the commonsense knowledge HellaSwag by [Zellers et al., 2019b] is chosen as a proxy downstream task. Additionally, GLUE is used to evaluate the effect of the retraining regime on the general language knowledge of the model.

Figure 3.3 depicts the complete flow of the experiments to be conducted.

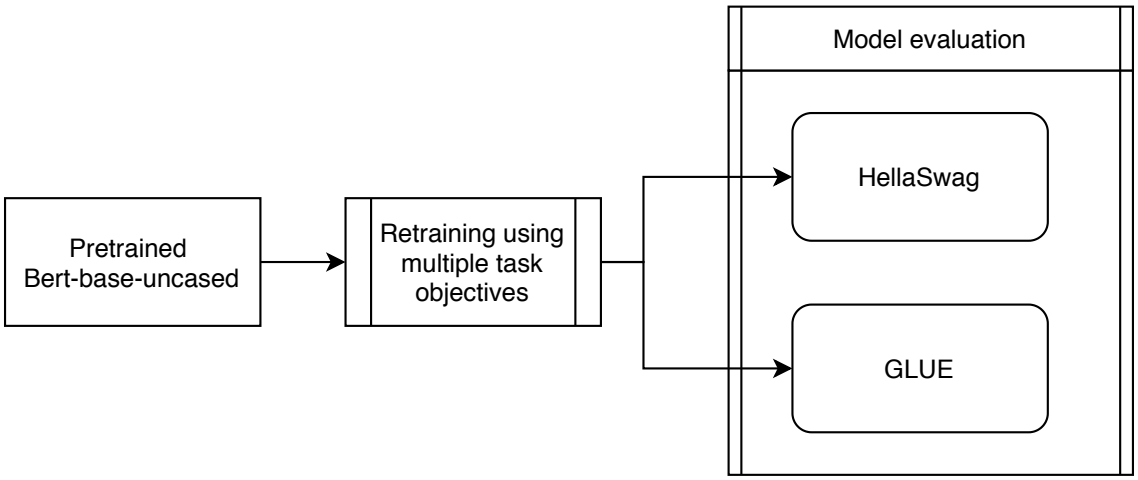

Figure 3.3: Experiment flow

Chapter 4

Implementation

4.1 Experiment environment

All the experiments are conducted in the computing cluster of the research group Data Science and Text-based Information Systems (DATEXIS) at Beuth University of Applied Sciences Berlin.

The technology choices of this work are in line with the ongoing research of the DATEXIS group. All the stages of the experiments are written in the python programming language and the PyTorch [Paszke et al., 2019] Deep Learning library. The main deployment mechanism in the cluster are Kubernetes jobs. The data preprocessing programs using CPU parallelism are implemented with the Ray library [Moritz et al., 2018]. All the model training experiments were done in a single NVIDIA V100 GPU with a batch size of 16 training examples per iteration. The finetuning downstream tasks were run on NVIDIA P100 and K80 GPUs.

4.2 Data

4.2.1 Sources

The ATOMIC dataset is retrieved from [ATOMIC, 2020] without any filtering, the CONCEPTNET data corresponds to the tuples from the Open Mind Common Sense in CONCEPTNET 5 [Speer and Havasi, 2012], enriched with additional edges and scores [Li et al., 2016].

4.2.2 Data Split

The split of the data for retraining is done as follows: 80% for training, 10% for the development set, i.e. controlling the training process, and finally, 10% for the testing. The precise number of examples of this split is shown in table 4.1.

As for the tasks of the retraining, each one of was weighted with roughly the same amount of examples. Table 4.2 shows the distribution of examples per task and set.

Split	# of Examples
Training	6419245
Development	802405
Test	802406

Table 4.1: Data split of the total number of examples.

Task	Train	Development	Test
IS_CORRECT	1069901	133670	133772
MASK_B	1070197	133406	133740
MC_NODEB	1069741	133916	133686
MC_NODEA	1069571	134328	133443
MASK_EDGE	1070183	133334	133826
MASK_A	1069652	133751	133939

Table 4.2: Number of examples per task and set.

Edge	# Examples
xAttr	114,856
xWant	104,132
xEffect	81,161
xNeed	77,496
xReact	62,725
xIntent	47,040
oWant	43,622
oEffect	28,443
oReact	26,492

Table 4.3: Edges in the training set of the ATOMIC knowledge base with their corresponding number of examples.

ATOMIC and CONCEPTNET distributions in training data

Tables 4.3 and 4.4 show the distribution of examples for ATOMIC and CONCEPTNET from the perspective of their edges. For a total of roughly 1.07 million examples 586K come from ATOMIC edges, and 484 from CONCEPTNET. The distribution in both bases is far from uniform in the edge dimension. This type of sampling bias is corrected at training time by balancing the loss proportionally for the edge prediction task.

4.2.3 Preprocessing

ATOMIC and CONCEPTNET are expressed as set of s, r, o triples. Given the large size of the merged data the preparation of the training, development and testing sets is done in advance, storing the results in a binary format to be preloaded at training. This processing involves the creation of the input data for BERT's forward pass, i.e. input tokens, targets, attention masks and token type ids.

Since the implemented model must abstract the input to all the tasks, every single data sample contains all the targets. However, with the exception of the relevant sample task, the targets are *masked out* i.e. their value is set to -100. Additionally, every sample also contains an identifier of the task so that when computing a prediction, the model assigns -100 to the *logits* that are not relevant for the loss, masking it out of the output heads that don't correspond to the task. With this mechanism only the loss gradients of the relevant task are propagated in the backward pass.

The BERT-base-uncased tokenizer from the HuggingFace library includes the original pretraining special tokens such as `[CLS]`, `[SEP]`, `[MASK]`, `[PAD]` among others. In order to make the tokenized input unambiguous for the retraining tasks, the additional special tokens

Edge	# Examples	Edge	# Examples	Edge	# Examples
UsedFor	68,264	CausesDesire	7,738	SymbolOf	260
CapableOf	57,865	Desires	7,187	RelatedTo	148
IsA	57,539	NotDesires	6,074	InstanceOf	115
AtLocation	56,860	HasFirstSubevent	5,877	InheritsFrom	88
HasSubevent	42,131	HasLastSubevent	4,725	NotMadeOf	39
HasPrerequisite	38,237	NotCapableOf	4,595	DesireOf	22
HasProperty	29,562	PartOf	4,459	LocationOfAction	5
Causes	27,964	NotHasProperty	1,489	LocatedNear	5
MotivatedByGoal	18,893	MadeOf	1,280	HasPainIntensity	2
ReceivesAction	16,539	NotIsA	782	HasPainCharacter	2
HasA	15,536	NotHasA	655		
DefinedAs	8,853	CreatedBy	426		

Table 4.4: Edges in the training set of the CONCEPTNET knowledge base with their corresponding number of examples.

Task	Input	Target
MASK_EDGE	[CLS] [NodeStart] **condor** [NodeEnd] [96X PAD] [EdgeStart] [8X MASK] [PAD] [NodeStart] **a bird** [NodeEnd] [SEP] [95X PAD]	IsA
IS_CORRECT	[CLS] [NodeStart] **baby** [NodeEnd] [96X PAD] [EdgeStart] **has ##a** [EdgeEnd] [6X PAD] [NodeStart] **soft skin than adult** [NodeEnd] [SEP] [93X PAD]	True

Table 4.5: Samples of input and targets for the triplet validation and edge prediction tasks

`[NodeStart]`, `[NodeEnd]`, `[EdgeStart]`, and `[EdgeEnd]` were added to the vocabulary and tokenizer. Table 4.5 shows a preprocessed tokenized example for the tasks of triplet validation (IS_CORRECT) and edge prediction (MASK_EDGE) the multi-task model.

4.3 Modelling

The multi-task model is implemented by extending PyTorch's `nn.Module` class. Additionally the BERT model used is the base-uncased variant from the HuggingFace transformers python library [Wolf et al., 2019]. As explained in chapter 3, the 4 tasks correspond to 6 output heads that are fed with BERT's last hidden layer. Listing 4.1 presents the member objects implementing such heads. The classification outputs, are created with an `nn.Linear` layer. These are:

- `self.output_heads_iscorrect` for binary classification in the triplet validation task.
- `self.output_heads_maskedge` Classifier of the triple edges.
- `self.output_heads_mc_nodea` Classifier over the multiple choice question answering task targeting node a or s.
- `self.output_heads_mc_nodeb` Classifier over the multiple choice question answering task targeting node b or o.

The reason why these layers are implemented only as linear instances in contrast with their definitions in chapter 3 as Softmax layers, is that the implementation of the Cross-entropy-loss in PyTorch is meant to be fed with logits, i.e. the output of a layer meant for classification before it has been mapped to a probability distribution. Internally this loss implementation applies the softmax function before computing the actual Cross-entropy loss [PyTorch, 2020].

The output heads of the masked language modelling tasks are implemented with the `BertOnlyMLMHead` layer from the HuggingFace library. This layer creates a `nn.Linear` layer spanning the hidden size of the last decoder layer of BERT with a width of the vocabulary size, i.e. modelling a distribution over BERT's vocabulary for each of the tokens of the output. This layer is also combined with a `CrossEntropyLoss` criterion.

Code Listing 4.1: MTLBert model members

```
    from torch import nn
    from transformers import BertModel, BertConfig, BertPreTrainedModel
    # Additional Imports
    # ...

    class MTLBert(nn.Module):
    def __init__( ... ):

        # Initialization code
        # ...
        self.criterions_cel      = nn.CrossEntropyLoss()

        self.bert = BertModel.from_pretrained(...)

        # Dropout layer and output heads

        self.dropout                      = nn.Dropout(dropout)

        self.output_heads_iscorrect   = nn.Linear(bertConfig.hidden_size, 2)
        self.output_heads_maskedge    = nn.Linear(bertConfig.hidden_size,
                                                  self.num_classes)
        self.output_heads_maska       = BertOnlyMLMHead(bertConfig)
        self.output_heads_maskb       = BertOnlyMLMHead(bertConfig)
        self.output_heads_mc_nodea    = nn.Linear(bertConfig.hidden_size,
                                                  self.num_choices)
        self.output_heads_mc_nodeb    = nn.Linear(bertConfig.hidden_size,
                                                  self.num_choices)

    # Additional class methods and definitions
    # ...
```

The forward computation of the `MTLBert` model is presented in listing 4.2. Initially the tokenized inputs, attention mask, token type ids and targets are passed through a BERT base-uncased model instance of the HuggingFace transformers library, then depending on the pooling strategies, the corresponding output or hidden state is passed through the dropout layer to subsequently be subjected to the loss function of each task.

In the sample of listing 4.2, the representation of the first token of the last hidden state, i.e. the `[CLS]` token, is passed through the dropout layer and it serves as the input for the classification layers.

The pooling is not applied for the masked language modelling tasks, instead the whole last hidden state of BERT is passed through the dropout layer before applying the loss.

Globally the total loss is simply computed by adding all the individual losses of each task.

Code Listing 4.2: Forward pass of MTLBert

```
    def forward( ... ):
        # Pass the triples through BERT first
        encoded = self.bert(...)

        # Pooling strategies
        # ...

        elif self.pooling_strategy == "CLS":
            pooled = encoded[1]
            pooled = self.dropout(pooled)
        # ...


        encoded = self.dropout(encoded[0])
        # ...
        logits['IS_CORRECT'] = self.output_heads_iscorrect(pooled)
        logits['MASK_EDGE']  = self.output_heads_maskedge(pooled)
        logits['MASK_A']     = self.output_heads_maska(encoded)
        logits['MASK_B']     = self.output_heads_maskb(encoded)
        logits['MC_NODEA']   = self.output_heads_mc_nodea(pooled)
        logits['MC_NODEB']   = self.output_heads_mc_nodeb(pooled)

        # Mask with -100 the logits that don't belong to the classes
        # ...

        # Compute the losses
        losses['IS_CORRECT'] = self.criterions_cel(logits['IS_CORRECT']...)
        losses['MASK_EDGE']  = self.criterions_cel_edge(logits['MASK_EDGE']...)
        losses['MASK_A']     = self.criterions_cel(logits['MASK_A']...)
        losses['MASK_B']     = self.criterions_cel(logits['MASK_B']...)
        losses['MC_NODEA']   = self.criterions_cel(logits['MC_NODEA']...)
        losses['MC_NODEB']   = self.criterions_cel(logits['MC_NODEB']...)

        # Accumulate the total model loss, and return it
        # ...
```

4.3.1 Retraining Routine and Hyper-Parameter Search

The retraining routine is run for 3 epochs over the whole data, lasting roughly between 30 to 80 hours depending on the combinations of data and tasks in each experiment iteration. Because of this retraining duration times an exhaustive hyper-parameter search is not conducted. Furthermore, such search would only be reasonable in orchestration with the downstream tasks, this renders the hyper-parameter space orders of magnitude larger. The problem of HPO can be tackled with the use of sophisticated algorithms and adequate software frameworks. However, this is not addressed in this work and left out as an open question.

An empirical iterative exploration of combinations of the datasets, tasks and dropout is conducted driven by the results of the common sense downstream task. This is discussed in Chapter 5.

Table 4.6 details the hyper-parameters used for the retraining process.

Hyper-parameter	Value
# of Epochs	3
Learning Rate	3e-5
Optimizer	ADAM
ADAM epsilon	1e-8
Dropout output heads	0.25 to 0.9
Dropout Hidden layers	0.1 to 0.8
Dropout Attention layers	0.1 to 0.8
BERT pooling for classification heads	[CLS] token

Table 4.6: Hyper-parameters used in the retraining.

4.4 Downstream tasks

Two downstream tasks were implemented to evaluate both the commonsense proficiency of the retrained model (HellaSwag) and the general language capabilities (GLUE).

4.4.1 HellaSwag

HellaSwag is an incremental step on SWAG, and involves a multi label classification problem where for each question a context with 4 different endings are given for the model to choose from. As such the implementation is identical to a SWAG implementation only with a more challenging dataset.

This work refactored an example contributed to the HuggingFace source repository, that focused on a multiple-choice question answering BERT model and SWAG. The main adaptions made correspond to the HellaSwag data acquisition from [Zellers et al., 2019a], preprocessing, and the model extraction from the `MTLBert` retrained binary with its corresponding modified tokenizer and configuration.

Every experiment was run on DATEXIS kubernetes cluster as a one-shot job with at most 3 random seeds for each experiment to keep the evaluation times tractable. On average on an NVIDIA P100 GPU every training and evaluation lasted roughly 6 hours.

The hyper-parameters chosen for this task are the same ones used by the BERT experiment of [Zellers et al., 2019b], this is to maintain a degree of comparability.

The test set of HellaSwag is unlabeled, this is because [Zellers et al., 2019b] keep a ranking. For the test set the evaluation of the examples must be submitted, and only then, the accuracies are computed and published in this scoreboard. All the results of this work for HellaSwag are computed on the evaluation set, both for the retrained models and the baseline. This is coherent with keeping the iterations in the experimental process within reasonable times.

4.4.2 GLUE

The implementation from the GLUE benchmark is fully enabled by the HuggingFace library, in this case the only overhead is to deploy each one of the 9 tasks as an individual job in the cluster. The total duration for the whole GLUE benchmark lasts roughly 30 hours on a single NVIDIA k80, even though the deployment of these jobs was heavily parallelized the benchmarks were run only for a single random seed for all the retrained models.

4.5 Summary

This chapter presents the implementation details of the experiments, targeting mainly the technical stack chosen to be in line with the resources of the DATEXIS research group. The implementation of the multi-task model uses heavily the PyTorch framework with the HuggingFace transformers library.

The data is first processed and stored in a binary feature file so that it can readily be used during the training routines.

The modelling simply involves the additional output heads that are fed with BERT's last hidden state given an input triplet. The classification tasks translate to 4 Linear layers which in combination with the `nn.CrossEntropyLoss` criterion from PyTorch fulfill the loss and softmax layers stated in Chapter 3. The masked language modelling tasks involve a HuggingFace `BertOnlyMLMHead` that is simply a linear layer with dimensions `hidden_size , vocab_size` that model a sequence of logits before a distribution over all the vocabulary that yet again is completed with the `nn.CrossEntropyLoss` criterion in the softmax sense.

Additionally, the forward pass implements the masking of the logits with -100 before the loss is applied and returned for the tasks that are not relevant for the example being processed. This guarantees that only the gradients of the loss corresponding to the example tasks are propagated in the backward pass.

The HellaSwag downstream task is a refactored SWAG example from the HuggingFace library adapted to the different format of the data from [Zellers et al., 2019a]. GLUE is fully implemented in HuggingFace and as such is run without major modifications.

Chapter 5

Results

5.1 Evaluation setup

5.1.1 Hypothesis

Given a pretrained model such as BERT-base-uncased, and a retraining with multiple tasks derived from commonsense knowledge bases, the retrained model will outperform a model of the same architecture without retraining, when put through a commonsense downstream task. Additionally, this retraining procedure will not affect significantly the capability of the model at general language tasks.

5.1.2 Baseline: Raw pretrained model

Since the main objective of this work is to validate the improvement of the performance of a retrained BERT-base-uncased on a commonsense downstream task like HellaSwag, the pretrained model without any additional retraining is taken as a baseline. For this, `bert-base-uncased` from HuggingFace with its default configuration is taken as a reference.

The hyper parameters chosen in the downstream task are the ones published in the source code for BERT-base of the HellaSwag paper [Zellers et al., 2019b], however given the fact that the implementation is different, in the sense of the machine learning frameworks used, the best results reproduced over 7 initial random states are the ones that are used as a reference. As for the retrained model no hyper parameter exploration is conducted, i.e. identical settings of the baseline are the only used.

The same logic is used when assessing the general language results. In this case the 9 tasks of GLUE are the downstream tasks and BERT-base-uncased without retraining is yet again the baseline.

5.1.3 Metrics

As presented in Chapter 2 the critical metric for HellaSwag is the Accuracy, i.e. the proportion of correct answers in the dataset, GLUE has various control metrics for the 9 different tasks, they are introduced extensively in Chapter 2.

5.2 Experimental evaluation

There are two different dimensions of the evaluation of the experiment, first comes the analysis of the retraining of BERT in the multitask setting, and subsequently the success of this retraining is

Task	Metric	Value
IS_CORRECT	F1-score	0.978000
MASK_EDGE	macro F1-score	0.569000
MC_NODEA	macro F1-score	0.894000
MC_NODEB	macro F1-score	0.949000
MASK_A	Perplexity	1.012000
MASK_B	Perplexity	1.013000
Multitask	Dev loss	11294.046875

Table 5.1: Evaluation set final retraining results

Task	Metric	Value
IS_CORRECT	F1-score	0.978000
MASK_EDGE	macro F1-score	0.567000
MC_NODEA	macro F1-score	0.895000
MC_NODEB	macro F1-score	0.950000
MASK_A test	Perplexity	1.011000
MASK_B test	Perplexity	1.013000
Multitask	Test loss	9517.152344

Table 5.2: Test set retraining results

only validated via the fine-tuning tasks.

5.2.1 Retraining evaluation

Tables 5.1 and 5.2 present the control metrics for the evaluation and test sets of a model that was retrained on the 6 tasks and both knowledge bases, namely the model tagged 2020-06-03_153757. The development metrics are taken during the training routine at intervals of 100K batches or alternatively 1.6M steps given that the batch-size for the training in this experiment was of size 16. The figures of Table5.1 are the last evaluation done during training.

For all tasks the retraining results show similar metrics for both test and development set.The F1-score is macro averaged for the classification with multiple classes to compensate for the uneven distribution of the labels. The model achieves high scores in the triplet validation and multiple choice tasks. In the latter, the fact that the prediction of the o node or the MC_NODEB consistently outperforms the other node (s or MC_NODEA), is possibly a consequence of the arbitrary assumption on the directionality in the relations in the graph triplets made in Chapter 3 not being true after all.

The triplet validation results (IS_CORRECT) are very consistent in both the development and test sets, with a very high F1-score of 0.97.

The edge-prediction task (MASK_EDGE) presents an F1-score of 0.56 which is not as high as the previous 3 tasks, but it is fairly above random chance for the 43 edges of the dataset.

Figure 5.1 combines the evolution of the development metric relevant to each task (green) with its corresponding training loss (blue). For visualization purposes the training loss has been smoothed with an exponential moving average.

In all of the above mentioned tasks the loss decreases steadily as the control F1-score metric increases with the training steps. This is consistent with a model that is learning. The training is limited to 3 epochs over the data to avoid over fitting. Nevertheless, the results of the

edge-prediction task might as well be explained by the fact that this task needs a longer retraining.

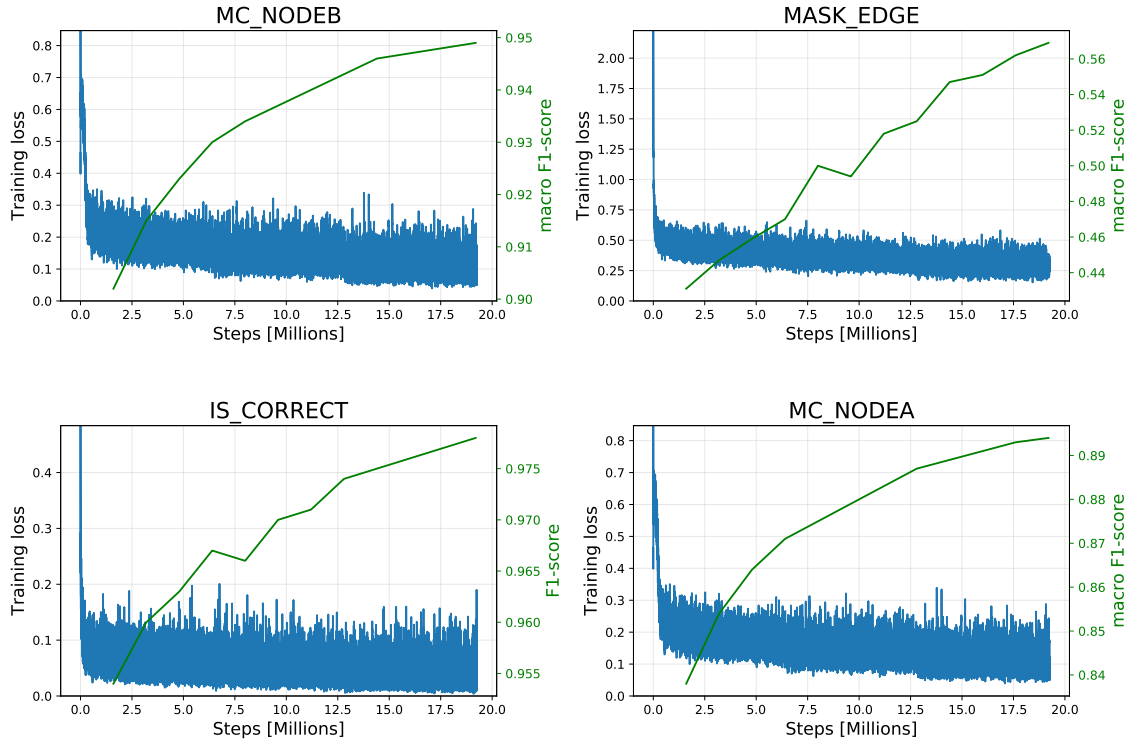

Figure 5.1: Multitask training results, in blue the loss of each task, in green the corresponding F1-score. The loss has been smoothed with an exponential, moving average for visualization purposes

The tasks of MLM need a deeper evaluation. The Perplexity for these tasks lays in the neighborhood of 1 which corresponds to a loss of approximately 0, since this metric is computed as the exponential function of the loss. Figure 5.2 shows the loss and the corresponding Perplexity for the MLM task when retraining this model. It is evident that the loss quickly drops around 0, suggesting either a problem with the formulation of the task or an unintended feature of the implementation.

To inspect what happens with the predictions and targets a few randomly chosen samples are examined for both tasks. Listing 5.1 shows a single of those samples for the MASK_A task. Although the prediction might seem fairly random at the beginning there's a very intricate issue in the way the problem is posed to the model. The input sequence consists of 210 tokens, for the MLM tasks 100 tokens correspond to each of the two nodes and 10 tokens are destined to the edges.

The input tokens for the model are constructed in a way that present multiple `[MASK]` tokens (to be completed), followed by the edge r and o node. The target presents the ground truth for the mask to be predicted padded until the start of the edge r and o node. The problem lies in the padding .i.e. the `[PAD]` sequences that are appended in the target after the ground truth. These are not masked out, so as long as the model predicts a significant amount of `[PAD]` tokens, for the initial 100 tokens, the total loss for that particular sample will be significantly low.

In Listing 5.1, it can be seen that the target has 90 `[PAD]` tokens following the ground truth within the initial 100 positions corresponding to s, and the model predicted 92 pads within the same neighborhood of tokens. Furthermore, the last 110 positions of the target are a -100 value

that intentionally is ignored in the implementation of the Cross-Entropy loss of PyTorch. This masks out the effect of the prediction of the last 110 tokens by BERT. The combination of these two aspects render the problem too trivial for the network (predict the padding) and explain the values of the loss and Perplexity. This behavior is consistent over the majority of the examples.

Originally the decision behind not masking out the padding in the targets was made so as not to hint the model with the total length of the sequence to be predicted. Nevertheless, this unwanted feature of the target construction was not discovered until fairly late in the experimental process, so a retraining with a different approach to the padding is left for further work. As a side effect the poor performance in this tasks is the reason behind further iterations of the retraining to exclude the two masked language modelling tasks.

Code Listing 5.1: MLM MASK_A inference example

```
Input :[CLS] [NodeStart] [97X MASK] [PAD]
             [EdgeStart] x ##wan ##t [EdgeEnd] [5X PAD]
             [NodeStart] to explain un ##im ##port ##ant things
                         to person y [NodeEnd] [SEP] [87X PAD]

Target :[CLS] [NodeStart] person ##x waste ##s person ##y time [NodeEnd]
              [90X PAD] [110X -100 MASK]

Prediction :[CLS] [NodeStart] person ##x gives person ##y ' s [92X PAD]
            [NodeEnd] [8X PAD] [NodeEnd] [2X PAD] [NodeEnd] [PAD] [NodeEnd]
            [3X PAD] [4X NodeEnd] [2X PAD] [NodeEnd] [12X PAD] [NodeEnd]
            [42X PAD] [NodeEnd] [PAD] [2X NodeEnd] [CLS] [2X NodeEnd]
            [CLS] [2X NodeEnd] [CLS] [NodeEnd] [4X PAD] [NodeEnd] [4X PAD]
            [NodeEnd] [4X PAD] [NodeEnd] [2X PAD]
```

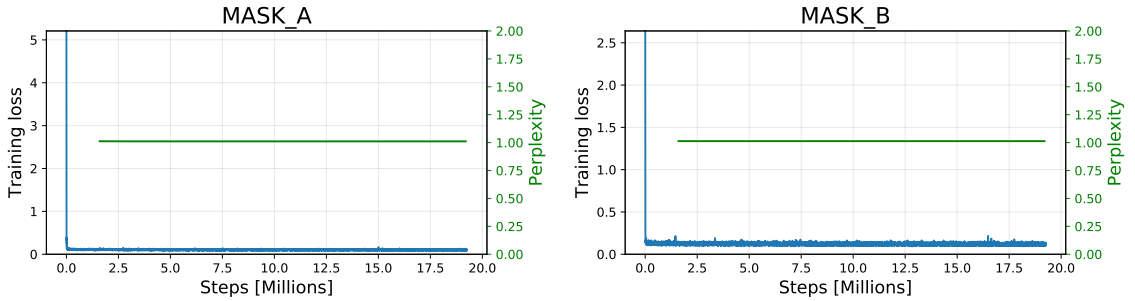

Figure 5.2: Retraining results of the MLM tasks. The training loss steeply converges to 0 stagnating the Perplexity in 1.

The overall Multitask loss for the retraining of this model is shown in Figure 5.3, this is simply an aggregation of the losses of all tasks that is dominated by the negative slope consistent with a model that is learning.

Iterations of the retraining were made with subsets of the data and the tasks, driven both by the retraining and downstream incremental results, these are discussed in more detail in the commonsense evaluation.

5.2.2 Commonsense evaluation

Given the duration of the retraining routines, only an empirical exploration of subsets of combinations of datasets and tasks is conducted for different retraining instances. Additionally, different settings of dropout are explored in the spirit of handling catastrophic

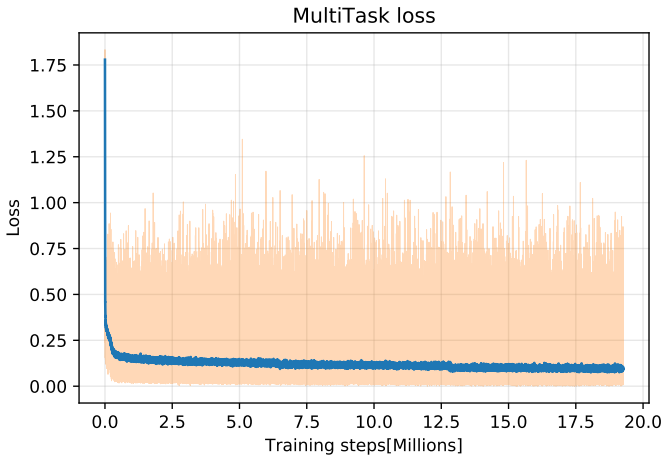

Figure 5.3: Multitask loss for the model 2020-06-03_153757, the loss(orange) is an aggregation of all the losses in the model, additionally an exponential moving average has been used to smoothen the curve (blue). Although slow, the negative slope of the curve is consistent with a model that is learning.

forgetting [Goodfellow et al., 2013].

The results of the HellaSwag downstream task are shown in table 5.3. The models are timestamped with the retraining initial time. There are 3 different dropout parameters that are considered in the retraining process:

- Dropout in the output heads, as illustrated in Chapter 4.
- Dropout in the attention layers of BERT.
- Dropout in the fully connected layers of BERT.

Since the retraining of the models takes a significant amount of time, the exploration of the dropout settings for these layers was limited mainly to edge cases i.e. high dropout on the Output heads to drive the propagation of the loss gradients mainly to the model, and also high dropout to BERT's hidden layers and attention to limit the extent of what can be learned.

The results show that none of the models is able to overcome the Accuracy of the baseline BERT-base, and in order to analyze whether the *forgetting* of the network could be targeted, ablations on the retraining were undertaken such as choosing only one knowledge base and removing the MLM tasks because of its poor retraining results.

The best performing retrained model (2020-06-23_144602) achieves 36.% of accuracy in comparison with the 38.3% of the baseline. This model was retrained with both CONCEPTNET and ATOMIC and all the tasks with the exception of the MLM tasks.

Furthermore, the retrained models that achieved the best results are the ones that have a very high dropout rate in the output layers. This suggests interestingly that the retraining has a less negative impact on the model once the gradient back propagation intuitively skips the output heads. In contrast, the single experiment run with higher dropout on the hidden and attention layers is the one with significantly worse accuracy in HellaSwag.

	Acc	Loss	Datasets	Tasks	O Dropout	H Dropout	A Dropout
2020-06-18_135229	25.5%	1.386295	[CONCEPTNET]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.5	0.8	0.8
2020-06-10_093127	33.6%	1.475318	[CONCEPTNET]	[MASK_A, MASK_B, MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.25	0.1	0.1
2020-06-03_153757	34.1%	1.448410	[CONCEPTNET, ATOMIC]	[MASK_A, MASK_B, MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.25	0.1	0.1
2020-06-26_191749	34.7%	1.462585	[CONCEPTNET]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.9	0.1	0.1
2020-06-11_041946	34.8%	1.535658	[CONCEPTNET, ATOMIC]	[MASK_A, MASK_B, MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.8	0.1	0.1
2020-06-09_072234	35.0%	1.517587	[CONCEPTNET, ATOMIC]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.25	0.1	0.1
2020-06-16_203240	35.4%	1.531923	[CONCEPTNET, ATOMIC]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.9	0.1	0.1
2020-06-16_154727	36.0%	1.489099	[ATOMIC]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.9	0.1	0.1
2020-06-15_120244	36.2%	1.570379	[CONCEPTNET, ATOMIC]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.8	0.1	0.1
2020-06-22_080804	36.5%	1.544366	[CONCEPTNET]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.25	0.1	0.1
2020-06-23_144602	36.7%	1.519188	[CONCEPTNET, ATOMIC]	[MASK_EDGE, IS_CORRECT, MC_NODEA, MC_NODEB]	0.9	0.1	0.1
bert-base-uncased	**38.3%**	2.044656	-	-	-	-	-

Table 5.3: Baseline and retrained model results on HellaSwag, **O Dropout**: Dropout applied to the multiple task output heads **H Dropout**: Dropout applied to the fully connected layers in the embeddings, encoder, and pooler. **A Dropout**: Dropout ratio for the attention.

task model_name	Mcorr CoLA	Acc MNLI	MRPC	F1 MRPC	Acc QNLI
2020-06-15-120244-42	0.46	0.83	0.77	0.85	0.89
2020-06-16-154727-673	0.38	0.81	0.73	0.83	0.88
2020-06-16-203240-673	0.37	0.81	0.76	0.84	0.88
2020-06-18-135229-673	0.00	0.35	0.68	0.81	0.51
2020-06-22-080804-673	0.48	0.82	0.78	0.85	0.90
2020-06-23-144602-673	0.41	0.81	0.80	0.86	0.88
2020-06-26-191749-141	0.26	0.80	0.78	0.85	0.87
bert-base-uncased	**0.57**	**0.85**	**0.86**	**0.90**	**0.91**

Table 5.4: Baseline and retrained results for the GLUE benchmark, tasks CoLA, MNLI, MRPC and QNLI

5.2.3 General language performance evaluation

The GLUE benchmark was used as a reference for the language evaluation. Tables 5.4 and 5.5 show the corresponding metrics for each one of the tasks of the retrained models and the baseline. Each model name is followed by a number representing the seed that achieved the best Accuracy in HellaSwag. Not all the models ran through HellaSwag were examined through the GLUE benchmark, only the best and worst performers.

In a very similar fashion to HellaSwag, the retrained models don't manage to overcome the raw pretrained BERT baseline with a few exceptions where they match it, and a single case where a retrained model surpasses it in the Winograd Schema Challenge WNLI.

The best performing retrained models on HellaSwag: 2020-06-15_120244, 2020-06-22_080804 and 2020-06-23_144602 show similar results as the baseline across all the GLUE tasks. These results are satisfactory in the sense that they don't show a significant degradation of the performance of the best retrained models.

The model that surpasses the baseline in WNLI (2020-06-18-135229) performs very poorly across other GLUE tasks and happens to be as well the model with the worst accuracy in HellaSwag. This model is also the one where the highest dropout rate is applied to the hidden and attention layers of BERT. This that throughout both GLUE and HellaSwag, the only dropout that makes a positive difference, is the one applied to the output heads.

5.3 Error Analysis and discussion

5.3.1 Quantitative Error Analysis

HellaSwag is a multiple choice answering task, the main evaluation metric being accuracy. Even though it is solved in a multi-class classification setup where each of the answers is a class, it does not make sense to analyze class-specific metrics because there is no independent meaning to each answer in terms of labels, i.e. every label happens to be the ordinal placement of the answer in a specific question, yet this order is arbitrary and conveys no semantic meaning. Instead, what can be done is to analyze the outcome of the downstream task in the sense of the Bias-Variance trade off.

For this purpose the train and evaluation errors are computed for the model based on the accuracy.

task Model Name	Acc QQP	F1 QQP	Acc RTE	SST-2	Pearson STS-B	Spearman STS-B	Acc WNLI
2020-06-15-120244	0.90	0.87	0.66	0.90	0.87	0.86	0.35
2020-06-16-154727-673	0.90	0.86	0.54	0.90	0.83	0.83	0.32
2020-06-16-203240-673	0.90	0.87	0.62	0.90	0.86	0.86	0.31
2020-06-18-135229-673	0.66	0.32	0.47	0.75	0.18	0.17	**0.56**
2020-06-22-080804-673	**0.91**	0.87	0.62	**0.92**	0.87	0.86	0.46
2020-06-23-144602-673	0.90	0.86	**0.65**	0.90	0.87	0.86	0.39
2020-06-26-191749-141	0.89	0.86	0.53	0.89	0.83	0.82	0.46
bert-base-uncased	**0.91**	**0.88**	**0.65**	**0.92**	**0.89**	**0.88**	0.46

Table 5.5: Baseline and retrained results for the GLUE benchmark, tasks QQP, RTE, SST-2, STS-B and WNLI

Model name	Tr. Acc	Ev. Acc	Tr. Error	Ev. Error	Bias	Variance
2020-06-23_144602	71.6%	36.7%	28.4%	63.3%	24.0%	34.9%
bert-base-uncased	88.1%	38.3%	11.9%	61.7%	7.5%	49.7%

Table 5.6: Train and test error metrics for best retrained model and baseline.

The best retrained model and the baseline are compared following these metrics in table 5.6. The training and evaluation error are computed as the difference to the 100% ideal accuracy from the corresponding data splits, the bias is the difference between the training error and the human error published on HellaSwag i.e. 4.4% (100% - 95.6 of accuracy % [Zellers et al., 2019b]), and the variance is the difference between the test and train error.

It can be seen that the retraining stage hinders the evaluation accuracy by roughly 1.6 percentage points, yet it regularizes the model by reducing roughly 14% from the variance. It is also evident that the baseline `bert-base-uncased` model shows a lower bias that corresponds with the over fitting to the training data, this can be explained with the fact that the hyper parameters of the downstream task are chosen to be explicitly the ones yielding the results of [Zellers et al., 2019b]. The exploration of the hyper-parameter space for HellaSwag and the retrained model remains as a potential path to improve the accuracy, however this is not pursued in this work and is left as an open question.

Table 5.7 shows a comparison between the best retrained model and the baseline. It shows the distribution of the correct and incorrect answers between the two models. It can be seen that the 1.6 percentage points of the degradation of the accuracy do not come uniquely from a lower capacity of the model to choose correctly. Instead, the retrained model chooses correctly in 1567 examples where the baseline fails, but fails in 1732 where the baseline guesses correctly. This difference of 165 examples (1732 - 1567) explains the decreased Accuracy.

In order to quantify the comparison of the two models an important analysis is to examine the Cross-entropy in the groups of examples of table 5.7. This is to characterize how far the models are from predicting the right answers. These groups of evaluation examples can be subdivided as follows:

1. Wins: Where the retrained model chooses the correct answer and the baseline does not (1567 examples).

	Baseline	
Retrained	Correct	Incorrect
Correct	2119	1567
Incorrect	1732	4624

Table 5.7: Comparison of the answers in the development set between the baseline raw pretrained BERT model and the retrained model 2020-06-23_144602

Model	Wins	Fails	Both right	Both wrong
Retrained	0.720902	1.772751	0.533695	2.102854
Baseline	2.502814	0.376710	0.255615	3.201260

Table 5.8: Comparison of the mean losses in the development set between the baseline raw pretrained BERT model and the retrained model 2020-06-23_144602. **Wins**: the retrained model chooses the correct answer and the baseline doesn't, **Fails**: the retrained model chooses the wrong answer while the baseline chooses the correct one. **Both right**: Both models choose correctly the answer and **Both Wrong**: Both models fail to choose a correct answer.

2. Loses: Where the retrained models chooses a wrong answer while the baseline chooses correctly (1732 examples).

3. Both right: Both models choose correctly the answer (2119 examples).

4. Both wrong: Both models fail to choose a correct answer (4624 examples).

The Cross-entropy of both models with respect to the correct predictions for these groups are averaged and presented in Table5.8.

Of critical importance are the predictions where the retrained model misclassifies the answer. Both in comparison with the baseline (Fails vs Wins), and where both models predict wrong answers, the retrained model presents a lower average Cross-entropy than the baseline. Similarly, in the contrary cases where the models are correct, the Cross-entropy of the retrained model is higher. This is consistent with predictions that in the *softmax* sense are less localized, i.e. on average the model is more *confused* about the predictions, hinting at over regularization or catastrophic forgetting. Some qualitative examples are presented next.

5.3.2 Qualitative Error Analysis

In line with the focus of the quantitative analysis, Listing 5.2 examines 3 random examples from the group of answers where the retrained model failed in contrast with the baseline (Fails). The listing features in the HellaSwag fashion a detokenized context or question, the label y, the softmaxed predictions of both models, and the detokenized endings.

From the English composition of the answers there is no clear reason or conclusion behind the choices of the retrained model. The only thing that is consistent with the quantitative analysis is the fact that the probabilities, in the softmax sense, predicted by the retrained model are more spread across all the choices.

Code Listing 5.2: Examples of the **Losses** group

```
Context : [CLS] a group of cheer ##leader ##s run onto a
                stage before a cheering audience.

y   Retrained   Baseline  Ending

1   0.19        0.82      they get into formation, then begin dancing and
                          flipping as male cheer ##leader ##s join them.

0   0.70        0.02      they perform a cheer routine before the girls,
                          along with makeup artists, spread out and pose.

0   0.01        0.13      they take turns jumping on each other like they
                          are performing karate.

0   0.10        0.02      they are then shown performing the type of
                          cheerleading dance, using baton ##s and pole vaults.


-------------------------------------------------------------------------------

Context : [CLS] a helicopter flies in some people who then start playing
                paint ##ball .

y   Retrained   Baseline  Ending

1   0.20        0.95      they run around obstacles and have a great time .


0   0.07        0.03      they lose their shirt during the fight and
                          run back the others.

0   0.28        0.01      they shoot at each other while the helicopter drone
                          who am looking on.

0   0.45        0.01      they chase a bird in the sky while others
                          walk around.


-------------------------------------------------------------------------------

Context : [CLS] the mother ins ##truct ##s them on how to brush their teeth
                while laughing.the boy helps his younger sister brush his teeth

y   Retrained   Baseline  Ending

0   0.02        0.02      she shows how to hit the mom and then kiss his dad
                          as well.

0   0.47        0.08      she brushes past the camera , looking better soon
                          after.

0   0.12        0.02      she glow ##s from the center of the camera as a
                          reaction.

1   0.39        0.88      she gets them some water to ga ##rg ##le in their
                          mouths.
```

5.4 Conclusions

The measure of success or failure of the retraining proposed in this work, lies in the ability to overcome the metrics of the raw pretrained BERT in the common sense task. In this regard the retraining proved to be **unsuccessful** albeit by a short margin. Consistently the results of the best retrained models in HellaSwag didn't see any significant performance degradation in GLUE.

Generally, the iterations of the experiments left important outcomes and lessons.

5.4.1 Relevance of dropout

A high dropout on the output heads proved to be a good step towards improving the results in the common sense task. It is probably the most puzzling and most promising result of the experiment. It hints at the fact that the output heads are only a mapping of the encoder to a task space, in an optimistic view this parameter can be seen as a gate or *knob* to inject knowledge into a model.

5.4.2 Hyper-parameters

Even though the dropout of the output heads, and combinations of the choice of datasets are hyper-parameters and were to a very modest extent explored, a deeper search for the best parameters in the combination of both the retraining and downstream tasks is desireable. Only inspecting corner cases of the dropout and different initialization seeds made a significant difference in this work.

5.4.3 Beyond retraining

Commonsense knowledge is of great interest for language models to become generally better. The choice of knowledge bases and retraining in this work assumes that the architectural capacity of BERT is sufficient to deal with learning commonsense. This assumption however is challenged by the results achieved in the experimental process, and either a more selective way of targeting the parameters, or an additional architectural component should be topics to consider. Further ideas are explored in Chapter 6.

Chapter 6

Summary and future work

6.1 Summary

The main goal of this work has been to explore an intermediate retraining routine as a means to improve the domain knowledge of transformer models. Commonsense knowledge is in many regards beneficial for the SOTA not only in NLP but in the wide spectrum of fields in Machine Learning. The main ingredients for the methodology of knowledge instillation proposed in this work are:

- a SOTA model, in this case BERT base.
- a domain specific downstream task, in this case HellaSwag.
- a domain specific knowledge base, in this case ATOMIC and CONCEPTNET.
- a multitask retraining routine.

The retraining routine involves a tasks rephrasing the knowledge base triples in a format compatible with sequence to sequence modelling. This work explores 4 different ideas expressed in 6 tasks:

- Triplet validation: given a sequence with triplet of s, r, o predict whether such triplet belongs to a knowledge base.
- Edge prediction: Given two nodes s, o predict the edge or relation r that connects them.
- Masked Language Modelling: Given a sequence of a node and and edge s, r predict a sequence corresponding to the correct node o. Similarly, predict a node s, given a sequence of relation and node r, o.
- Multiple Choice: choose from 4 alternative endings including a correct node s (or o) given the corresponding node o (or s) and an edge r

The retraining is controlled with the referent metrics for each one of the tasks, for the triplet validation, Edge prediction and Multiple Choice a common classification metric such as the F1-score is chosen. For the MLM tasks the Perplexity is used.

Results of the retraining include satisfactory learning metrics on all the tasks with the exception of the MLM tasks. The latter were incrementally excluded from the retraining iterations since they showed a poor performance. These iterations consisted on the full evaluation of all the downstream tasks, HellaSwag and GLUE, after each retraining experiment.

The results of these downstream tasks show that the retraining procedure is **not** successful at instilling commonsense to the extent of overcoming the metrics of the baseline, a raw pretrained BERT model, in both HellaSwag and GLUE. Hence rejecting the hypothesis stated in Chapter 5. However, the metrics achieved are very close, and dropout on the additional output heads of the model that are attached for retraining, proved to be a critical factor in this.

The quantitative error evaluation suggests the potential in improving these results with HPO, but given the fact that such a procedure would involve combinations of hyper-parameters of both the retraining and downstream tasks, this idea was not explored as it is computationally beyond the scope of this work.

6.2 Future work

The following sections elaborates on further steps in the direction of perfecting the methodology proposed in this work.

6.2.1 MLM Tasks

Chapter 5 exposes a problem in the way the MLM task is formulated when introducing the padding of the node sequences. If the assumption of not hinting the model with the length of the nodes s and o is disregarded, and the padding is masked, the task can be rephrased in a non trivial way that can potentially infuse knowledge in BERT. [Wang et al., 2018a] emphasize on how important the original MLM task is for yielding good pretrained BERT models. This highlights the relevance of achieving better results in this task.

[Joshi et al., 2020] Show an alternative to the MLM task by replacing its loss with a so called Span Boundary Objective, targeting spans of sequences instead of individual tokens like in the original pretraining of BERT or the MLM tasks of this work. Exploring this replacement of the loss is also a step towards improving this task.

6.2.2 Selective propagation of the loss

Finding criteria to select the *redundancy* of the parameters of BERT like in [Michel et al., 2019], could be a good step into selecting which sets of parameters to target when retraining the model. This could be seen as finding sub architectures in BERT that can uniquely condense the knowledge bases.

6.2.3 Hyper-parameter tuning

The spirit of this work was to abstain from HPO because not only it involves a significant computation overhead, but also doing so was in favor of comparability with the baseline. Nevertheless, given the significant difference that a single parameter like the output heads dropout made in the experiments, HPO is most certainly an avenue towards improving on the results of this work.

6.2.4 Domain specific knowledge bases

The application of the methodology proposed could be explored in the context of very niche knowledge bases with smaller datasets. This in combination with different downstream tasks, for instance, in the medical domain could yield positive results where *small* data is an issue.

6.2.5 Further Models

Given how rapid has been the evolution of the SOTA, BERT is by far no single reference, applying the retraining to other language models should shed light on further improvements on the methodology.

References

[ATOMIC, 2020] ATOMIC (2020). Atomic an atlas of machine commonsense for if-then reasoning. https://homes.cs.washington.edu/~msap/atomic/. Accessed July 13, 2020.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bosselut et al., 2019] Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Celikyilmaz, A., and Choi, Y. (2019). Comet: Commonsense transformers for automatic knowledge graph construction. *arXiv preprint arXiv:1906.05317*.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

[Chen et al., 2016] Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H., and Inkpen, D. (2016). Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[ConceptNet5, 2020] ConceptNet5 (2020). Bicycle, an english term in conceptnet 5.8. https://conceptnet.io/c/en/bicycle. Accessed July 7, 2020.

[Davis and Marcus, 2015] Davis, E. and Marcus, G. (2015). Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM*, 58(9):92–103.

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[Garg et al., 2019] Garg, S., Vu, T., and Moschitti, A. (2019). Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. *arXiv preprint arXiv:1911.04118*.

[Gers et al., 1999] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[Goodfellow et al., 2013] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

[Gordon and Van Durme, 2013] Gordon, J. and Van Durme, B. (2013). Reporting bias and knowledge acquisition. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, page 25–30, New York, NY, USA. Association for Computing Machinery.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[Gururangan et al., 2018] Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. R., and Smith, N. A. (2018). Annotation artifacts in natural language inference data. *arXiv preprint arXiv:1803.02324*.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

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

[Joshi et al., 2020] Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., and Levy, O. (2020). Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

[Joulin et al., 2017] Joulin, A., Grave, E., Bojanowski, P., Nickel, M., and Mikolov, T. (2017). Fast linear model for knowledge graph embeddings. *arXiv preprint arXiv:1710.10881*.

[Li et al., 2016] Li, X., Taheri, A., Tu, L., and Gimpel, K. (2016). Commonsense knowledge base completion. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1445–1455.

[Marcus, 2018] Marcus, G. (2018). Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*.

[McCloskey and Cohen, 1989] McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

[Michel et al., 2019] Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024.

[Moritz et al., 2018] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. (2018). Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Peters et al., 2019] Peters, M. E., Neumann, M., Logan IV, R. L., Schwartz, R., Joshi, V., Singh, S., and Smith, N. A. (2019). Knowledge enhanced contextual word representations. *arXiv preprint arXiv:1909.04164*.

[Phang et al., 2018] Phang, J., Févry, T., and Bowman, S. R. (2018). Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

[PyTorch, 2020] PyTorch (2020). Cross entropy loss. `https://pytorch.org/docs/master/generated/torch.nn.CrossEntropyLoss.html`. Accessed June 9, 2020.

[Sanh et al., 2019] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

[Sap et al., 2019] Sap, M., Le Bras, R., Allaway, E., Bhagavatula, C., Lourie, N., Rashkin, H., Roof, B., Smith, N. A., and Choi, Y. (2019). Atomic: An atlas of machine commonsense for if-then reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3027–3035.

[Siegelmann and Sontag, 1992] Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449.

[Speer et al., 2017] Speer, R., Chin, J., and Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[Speer and Havasi, 2012] Speer, R. and Havasi, C. (2012). Representing general relational knowledge in conceptnet 5. In *LREC*, pages 3679–3686.

[Sutskever et al., 2011] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *ICML*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[Wang et al., 2018a] Wang, A., Hula, J., Xia, P., Pappagari, R., McCoy, R. T., Patel, R., Kim, N., Tenney, I., Huang, Y., Yu, K., et al. (2018a). Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. *arXiv preprint arXiv:1812.10860*.

[Wang et al., 2018b] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018b). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

[Wang et al., 2020] Wang, R., Tang, D., Duan, N., Wei, Z., Huang, X., Cao, C., Jiang, D., Zhou, M., et al. (2020). K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*.

[Warstadt et al., 2019] Warstadt, A., Singh, A., and Bowman, S. R. (2019). Cola: The corpus of linguistic acceptability (with added annotations).

[Wolf et al., 2019] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

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

[Zellers et al., 2019a] Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019a). Data. `https://github.com/rowanz/hellaswag/tree/master/data#data`. Accessed July 7, 2020.

[Zellers et al., 2019b] Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019b). Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.