City, University of London | MSc in Data Science

Project Report | 2019

ParaPhrasee: Paraphrase Generation using Deep Reinforcement Learning

phrasee

Artificial intelligence.
Human language.
Awesome.

CITY UNIVERSITY
LONDON

In partnership with Phrasee

Andrew Gibbs-Bravo

Supervised by: Eduardo Alonso

December 16, 2019

Declaration

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Acknowledgements

I would like to thank Neil, Elena, and the rest of the Phrasee team for providing valuable discussion throughout the development of the project and for sponsoring the project. I would also like to thank Dr. Eduardo Alonso for supervising the work and offering very useful guidance about technical questions and general feedback. Finally, my girlfriend Annie for her support and help throughout the many ups and downs in the research process.

Abstract

Automatically generating high quality paraphrases is a key problem for many tasks in Natural Language Processing and it also contains many important subproblems within it. While certain advances have been made through applying neural networks, this project explores using deep reinforcement learning to improve paraphrase generation quality. Various state-of-the-art supervised approaches to text generation are first compared and their shortcomings examined. Several common strategies for knowledge transfer between the supervised model and reinforcement learning model are then evaluated through two simple intermediary environments. Finally, we evaluate the impact of fine-tuning using reinforcement learning with different reward functions on paraphrase generation quality. We show the significant challenges in designing reward functions for paraphrase generation and that the best reward function is in fact using an adversarial model. We also propose a general strategy for transferring information from supervised models to reinforcement learning models to improve the efficiency of training. This work has significant implications in terms of not only improving paraphrase generation, but also proposing a universal pipeline that others can use when applying reinforcement learning to Natural Language Processing.

Keywords: Natural Language Generation (NLG), Metrics for Automatic Language Evaluation, Deep Reinforcement Learning (DRL), Actor-Critic Algorithm (A2C), Adversarial Training

Word Count

- Content Including Headers & Titles: 22,675
- Appendices: 22,046

Table of Contents

1 Introduction

Language is an incredibly beautiful and complex system and is in many regards the pinnacle of human achievement. The ability to communicate both real and imaginary ideas, context, and ultimately meaning to others through the use of symbols is an incredible feat which has resulted in substantial human evolution. An even more impressive attribute of language is that a single idea can be represented and communicated using a large variety of different symbols. One of the important considerations of language is that no two people interpret the same sentence in an identical way and as such, the specific choice of language will result in achieving different levels of understanding and response in the reader. This project addresses the generation of paraphrases at the sentence level.

What is a sentential paraphrase?

For the purposes of this project, a sentential paraphrase is a paraphrase at the sentence level which satisfies two important requirements:

- **Semantic similarity:** the two sentences must convey the same meaning. This means that they could be used roughly interchangeably to communicate the same idea to another person.
- **Fluency:** the generated sentence must be "fluent". This means that independent of how much semantic similarity the generated sentence shares with the source sentence, if a native English-speaking person read the generated sentence, they would be convinced that a human had written it (Gatt et al., 2018).

Automatically generating high quality sentential paraphrases remains a challenging problem which contains within it many subproblems that are at the core of the field of Natural Language Processing (NLP). Successful paraphrase generation requires achieving a good understanding of sentence pair modelling, language understanding, and language generation and has very broad applications including information retrieval, chatbots, translation, and text summarization (Prakash et al., 2016).

One of the problems with the current dominant approaches to text generation is models operate at the local level trying to correctly predict the next word without considering the generated sentence as a whole. The field of Reinforcement Learning (RL) defines a class of approaches to solving sequential problems to get the highest possible long-term reward (called return). This enables models to consider the global performance across the sentence and achieve improved generation quality (Ranzato et al., 2015). The generation procedure is similar between models: the model has a representation of the input sentence and the words which have already been predicted and based on this it predicts the subsequent word. The representation is then updated based on the prediction and the model is asked to predict the next word in the sequence.

RL has experienced a significant amount of growth as a result of recent success from applying deep neural networks to challenging problems. While RL algorithms and techniques can be used to perform well in complex environments such as Go (Silver et al., 2016; Silver et al., 2017), Atari (Mnih et al., 2013), and StarCraft II (DeepMind, 2019) they have not been as widely applied to NLP problems (with the exception of dialogue systems).

While predicting the next word in a sequence is technically a multiclass classification problem, evaluating the performance of the generative model is very challenging and remains an open problem with commonly used metrics showing significant variation from human judgment (Novikova et al., 2017; Liu et al., 2017; Vedantam et al., 2015; Anderson et al., 2016). This is a very significant problem in paraphrase generation as paraphrase identification itself remains an open problem and paraphrase quality is particularly subjective (Rus et al., 2011).

As a result, we show that optimizing on conventional metrics results in poor generation quality and instead propose using an adversarial approach to generation using a deep reinforcement learning model named ParaPhrasee. The model achieves strong performance on the paraphrase generation task without having to manually create a reward function. The approach can also be extended to achieve controllable generation through adding auxiliary reward functions.

Overview of Phrasee

Phrasee is a short-form text generation company which focuses on optimizing marketing copy. Phrasee's flagship product generates subject lines for email marketing campaigns. Phrasee has developed an in-house natural language generation system to automatically generate multiple variations of suitable language. They use deep learning to predict which language variations will achieve strong performance (e.g. high email open rates). This work investigates extensions and alternatives for the existing NLG system.

1.1 Research Question

Given this project exists in the intersection of multiple fields a primary research question was developed in addition to several supporting sub-questions.

Primary Question: What is the most effective reinforcement learning reward function for paraphrase generation?

Secondary Questions

- What is the best sequence to sequence architecture for paraphrase generation?
- How can knowledge obtained through supervised learning be leveraged to decrease computation requirements and training time for RL agents?
- How does performance vary between maximum likelihood estimation supervised training and reinforcement learning objectives?
- What is the impact of using Monte Carlo Tree Search on performance for trained reinforcement learning models?

1.2 Aim and Objectives

The aim of this project is to design a pipeline Phrasee can use to generate high quality sentential paraphrases as part of their subject line generation pipeline. In order to accomplish this aim, several project objectives have been defined as follows:

Objective	Testable Result
Thorough literature review of approaches to natural language generation, particularly paraphrase generation Thorough literature review of existing automatic evaluation methods	Comprehensive “Context” section in final project report
Evaluate performance of different encoder models on supervised paraphrase generation	Results comparing performance across numerous metrics
Trained GRU model on paraphrase sentence pairs	Trained paraphrase generation model using a specified encoder and decoder model
Fine-tuned RL language model / decoder on specified objective (reward function)	Trained RL paraphrase generation model fine-tuned model on reward resulting in improved performance
Identify best existing metric for evaluating performance in paraphrase generation	Use of principled metric with theoretical justification to assess model performance
Contribute to “world’s body of knowledge” (Dawson, 2009, p. 17)	Research demonstrating the best approach in applying reinforcement learning to fine-tune paraphrase generation models

Develop GUI / tool Phrasee can use to generate sentential paraphrases	Tool in which short form text is entered and optimized text is returned with the same semantic meaning
Develop list of future projects and extensions which build off this work	Comprehensive ‘Future Works’ section in final project report

1.3 Work Products

The project is intended to deliver the following products:

- A tool for Phrasee to create better performing short form text. Given an input sentence the model returns a high-quality paraphrase in terms of semantic similarity and fluency.
- A comparison of supervised and RL approaches to paraphrase generation.
- A principled approach that practitioners wanting to use RL for natural language generation can follow.

1.4 Project Beneficiaries

The project beneficiaries can be thought of in terms of three broad groups.

- **Phrasee:** The main project beneficiary is Phrasee whose main business is generating and evaluating high performing subject lines and other short form text. A model which could improve existing approaches would significantly benefit Phrasee’s clients and generate substantial revenue.

- **NLP community:** Given the broad applicability of generating paraphrases there are many NLP problems which would benefit from improvements in paraphrase generation approaches. Specific examples include:
    - Information retrieval (Culicover, 1968)
    - Chatbots (Li et al., 2019)
    - Question-answering (Dong et al., 2017)
    - Summarization (Paulus et al., 2017)

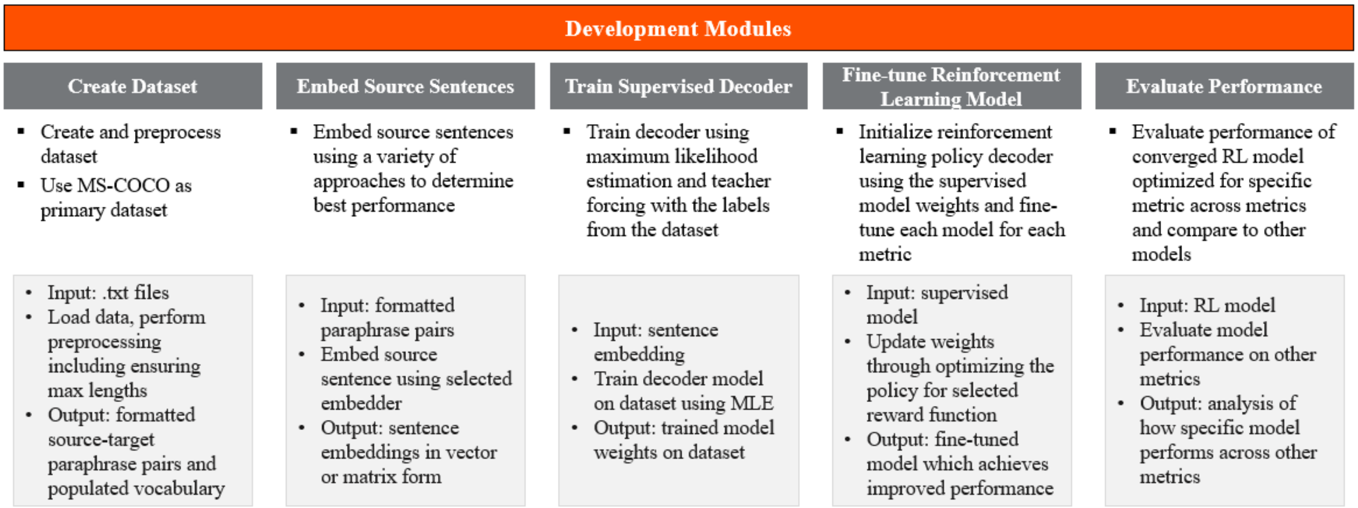
- **RL community:** While RL has been very successful in many domains, it has been underapplied to language problems. This is partially due to the fact that language has an immense state space which is problematic for many RL algorithms. This project seeks to “contribute to the world’s body of knowledge” (Dawson, 2009, p. 17) by developing a framework, other researchers can follow to apply RL agents to NLP problems.

1.5 *Methods Outline and Project Plan*

As this project involved a reasonable level of computational resources, resulted in a tangible work product, and needed to remain flexible to changes as better approaches were discovered, an iterative software model was the most appropriate. IBM's Rational Unified Process (RUP) outlines four phases of development and engineering workflows with building blocks. The phases are as follows:

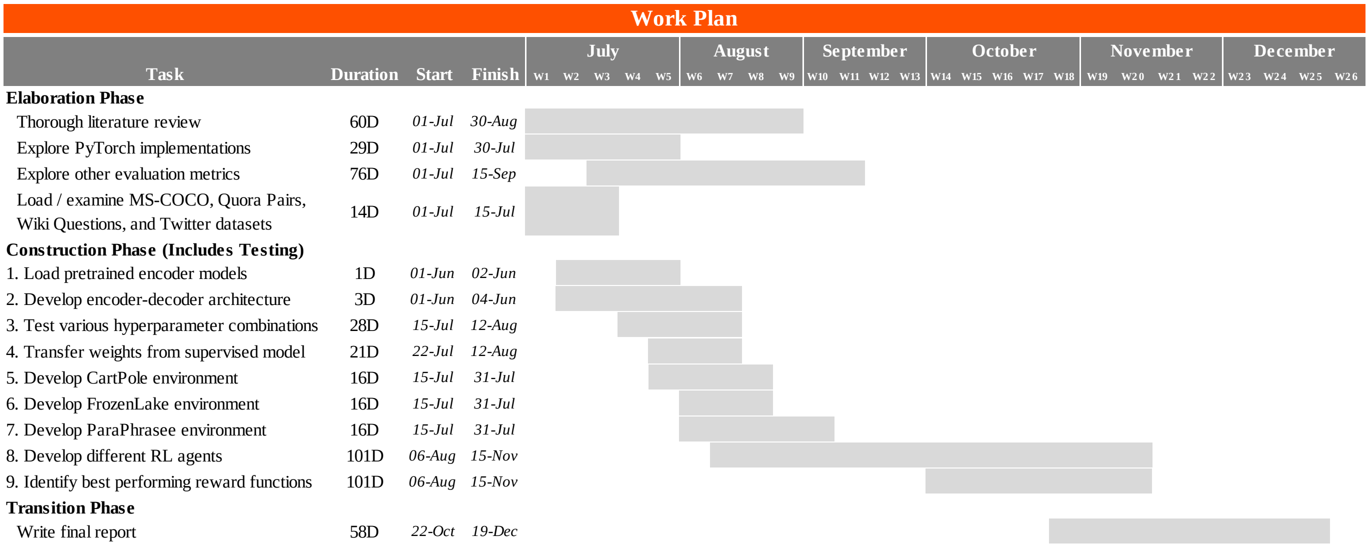
- ***Inception Phase:*** outlines the project feasibility and high-level requirements.
- ***Elaboration Phase:*** serves to further analyze and refine the requirements and will include a thorough literature review and increasing familiarization with existing Python implementations and datasets.
- ***Construction Phase:*** coding and implementation. The majority of the time was spent in construction.
- ***Transition Phase:*** where the final product is released and delivered to Phrasee and a maintenance plan in created. The plan is not to deploy the model explicitly but rather to extract the key insights for later integration.

The development modules and work plan are outlined below which are expanded upon in section 3.



Development Modules

Create Dataset	Embed Source Sentences	Train Supervised Decoder	Fine-tune Reinforcement Learning Model	Evaluate Performance
▪ Create and preprocess dataset ▪ Use MS-COCO as primary dataset	▪ Embed source sentences using a variety of approaches to determine best performance	▪ Train decoder using maximum likelihood estimation and teacher forcing with the labels from the dataset	▪ Initialize reinforcement learning policy decoder using the supervised model weights and fine-tune each model for each metric	▪ Evaluate performance of converged RL model optimized for specific metric across metrics and compare to other models
• Input: .txt files • Load data, perform preprocessing including ensuring max lengths • Output: formatted source-target paraphrase pairs and populated vocabulary	• Input: formatted paraphrase pairs • Embed source sentence using selected embedder • Output: sentence embeddings in vector or matrix form	• Input: sentence embedding • Train decoder model on dataset using MLE • Output: trained model weights on dataset	• Input: supervised model • Update weights through optimizing the policy for selected reward function • Output: fine-tuned model which achieves improved performance	• Input: RL model • Evaluate model performance on other metrics • Output: analysis of how specific model performs across other metrics



Work Plan

Task	Duration	Start	Finish
Elaboration Phase			
Thorough literature review	60D	*01-Jul*	*30-Aug*
Explore PyTorch implementations	29D	*01-Jul*	*30-Jul*
Explore other evaluation metrics	76D	*01-Jul*	*15-Sep*
Load / examine MS-COCO, Quora Pairs, Wiki Questions, and Twitter datasets	14D	*01-Jul*	*15-Jul*
Construction Phase (Includes Testing)			
1. Load pretrained encoder models	1D	*01-Jun*	*02-Jun*
2. Develop encoder-decoder architecture	3D	*01-Jun*	*04-Jun*
3. Test various hyperparameter combinations	28D	*15-Jul*	*12-Aug*
4. Transfer weights from supervised model	21D	*22-Jul*	*12-Aug*
5. Develop CartPole environment	16D	*15-Jul*	*31-Jul*
6. Develop FrozenLake environment	16D	*15-Jul*	*31-Jul*
7. Develop ParaPhrasee environment	16D	*15-Jul*	*31-Jul*
8. Develop different RL agents	101D	*06-Aug*	*15-Nov*
9. Identify best performing reward functions	101D	*06-Aug*	*15-Nov*
Transition Phase			
Write final report	58D	*22-Oct*	*19-Dec*

An illustrative work plan is above which outlines the approximate allocation of time spent across the project.

1.6 Changes During the Project

Given the significant amount of research conducted as part of developing the initial proposal, there were no fundamental changes in the aims or objectives of the project. In terms of achieving the objectives, there were numerous refinements made in the course of trial and error and as part of conducting the literature review. One of the main limitations in approach versus other state-of-the-art approaches in RL is the amount of compute required which is in our view deep reinforcement learning's "dirty little secret". Many high-quality papers either grossly underemphasize the amount of training time required or do not report training times.

As a result of the long training times and instability in convergence relative to supervised learning, intermediate environments were created with toy problems in order to debug algorithms and determine training and model transfer strategies that achieved consistent performance improvements.

A specific architectural change to the initial proposal is the use of an actor-critic reinforcement learning architecture rather than relying on the previously planned REINFORCE algorithm. The improved learning and stability through using actor critic resulted in a dramatic difference in convergence and training properties.

1.7 Report Outline

The remainder of the report is structured as follows:

- Chapter 2 – **Context**: Provides an overview of prior work across four of the key research areas most related to the project and how they have been applied to ParaPhrasee. These consist of:
    - **Natural language generation:** training models to generate unconditional or conditional text which appears human generated often to achieve another objective (e.g. translation, text summarization, chatbots, story generation, poetry and art generation, etc.).
    - **Automatic evaluation metrics:** developing metrics to approximate human-level evaluation of the performance of generated text.
    - **Deep reinforcement learning:** using neural networks to build models that can optimize long term non-differentiable rewards and scale to high dimensional problems.
    - **Paraphrase generation:** building systems which given input text generate output text which holds the same semantic meaning and is grammatical (although not necessarily fluent as will be discussed).
- Chapter 3 – **Methods**: Provides a detailed explanation of the approaches used in answering the research question including selecting and preprocessing the data, building the supervised model architectures including testing different encoder models, reinforcement learning architectures, and reinforcement learning environments.
- Chapter 4 – **Results**: Discussion of what the results from applying the methods were, the critical findings, and a high-level ablation study discussing what the key components were in achieving the results.
- Chapter 5 – **Discussion**: Evaluates the results relative to the project objectives, within the wider perspective of other related work, and its implications for other research areas. This section will also discuss how successfully ParaPhrasee achieves the initial objectives set out. Finally, it will cover the scope, generalizability, and validity of the findings including challenging the assumptions embedded in the approach and its resulting limitations in practice.

- Chapter 6 – **Evaluation, Reflections, and Conclusions**: Discusses and evaluates the project as a whole including what was achieved relative to the proposal. Reflecting on the design choices and what changes we would make if we were to restart and rescope the project. This section will also cover a comprehensive future works section with potential technical improvements in addition to broader philosophical changes which would allow the work to extend beyond paraphrase generation.

2 Context

This project sits at the intersection of two major research areas: NLP and RL. Therefore, before considering the context surrounding prior approaches to specifically paraphrase generation, it is worth reviewing the context to four distinct subproblems: natural language generation (NLG), automatic evaluation metrics, deep reinforcement learning (DRL), and finally paraphrase generation. For each subproblem we will review the traditional approaches, state-of-the-art tools, and how prior work has been used to develop ParaPhrasee.

2.1 Natural Language Generation (NLG)

Natural language generation addresses the problem of generating text, often to achieve a specific goal or for a specific audience. McDonald (2010) characterizes NLG as "the process by which thought is rendered into language" which is the view we share for this project although with a focus on "data" instead of thought. There are many applications of NLG including: translation, summarization, next word prediction, dialogue / chatbots, etc. Most useful applications focus on conditional generation where the model is given a representation of an input which influences the text it generates. Examples of input include image captioning: where given an image the model needs to generate an appropriate caption, report generation: where given a table of structured information (e.g. the weather) the model needs to generate coherent text conveying the information, or translation where given an input sentence in English the model needs to generate a corresponding sentence in French.

Traditional Approaches

Traditional approaches to language generation are mostly rule-based and rely on constructing templates which would be filled based on the context (Gatt et al., 2018). One of the main advantages of this approach is as long as the rules and templates have been configured correctly, the generated output is guaranteed to be grammatical. Rule-based approaches also give the users a high degree of control in generation. Template approaches are however very manual, and it is often intractable to capture all the desired behaviour in a rule set (Kondadadi et al., 2013). Statistical approaches to generation have long been considered a promising avenue of NLG and specifically having a single model responsible for natural language understanding (NLU) and NLG. However, such systems (e.g. bidirectional grammars) were challenging to build in practice (Reiter & Dale, 2000). As a result, most traditional natural language generation consisted of three components: a text planner, sentence planner, and realiser (Gatt et al., 2018). The success of applying neural networks has revolutionized many fields including NLP. As a result, the majority of contemporary state-of-the-art approaches rely on statistical learning leveraging neural networks and have moved towards integrated approaches to language generation rather than breaking the task into subcomponents (PapersWithCode, 2019; Gatt et al., 2018).

Neural Approaches

The application of neural networks to many problems in high dimensional space has proven extremely effective (Goodfellow et al., 2016). Kukich applied neural networks to NLG dating back to 1987 although limitations in hardware and overhyped results led to relatively little subsequent research into applying neural networks (Goodfellow et al., 2016).

Language modelling addresses the problem of determining the probability of the next word given the prior sequence of words and can be used for statistical generation through sampling the distribution. Naïve approaches to language modelling include using Markov Chains or other simple Bayesian conditional probabilities. These approaches are limited due to the high dimensionality of the possible combinations of words and limited ability to capture dependencies greater than several words. The state-of-the-art approaches rely on neural networks to represent the prior words in the sequence and predict the conditional probabilities of the next word.

Sutskever et al. (2014) extend the use of neural networks as a language model (Bengio et al., 2003) to end-to-end machine translation through the introduction of the encoder-decoder framework (also referred to as a seq-to-seq architecture). While the idea of applying recurrent neural networks to translation was proposed earlier by Kalchbrenner & Blunsom (2013), Sutskever et al. improved the architecture and were able to beat phrase-based translation approaches. This further revolutionized the field of NLP as handling sequences elegantly while achieving high performance had been a major hurdle to many tasks. Encoder-decoder architectures are now extremely common (Dusek et al., 2018).

A further improvement was introduced by extending the use of attention (Bahdanau et al., 2014, Kim et al., 2017) to create the Transformer model (Vaswani et al., 2017) which instead of relying on recurrence as required by Recurrent Neural Networks (RNNs) such as LSTMs and GRUs, uses an attention mechanism to model global dependencies between input and output pairs. This architecture achieved state-of-the-art in machine translation and is much faster than recurrent approaches as it can be easily parallelized (Vaswani et al., 2017). Attention and self-attention paved the way for later architectures to scale up the concept and achieve state-of-the-art performance such as GPT2 (Radford et al., 2019), BERT (Devlin et al., 2018), XLNet (Yang et al., 2019), and many others. A surprising outcome of improved performance in language modelling is how much linguistic information is captured by the model as part of its training. A consequence of this is that models trained on the language modelling task can be used as feature extractors across different NLP tasks.

Application to ParaPhrasee

The prior work in NLG was primarily used to inform the overall project structure such as modelling the problem using an integrated statistical approach rather than a rule-based system consisting of a text planner, sentence planner, and realiser. The baseline supervised model is an encoder-decoder architecture using a GRU. We also tested BERT, InferSent, GloVe embeddings, and attention models based on the success of these models in their respective published results.

2.2 Automatic Evaluation Metrics

In contrast with most problems in machine learning, the largest problem in most NLG applications does not lie in the modelling but rather in the evaluation. While human evaluation is currently often considered the gold standard in NLG evaluation, it suffers from significant disadvantages including being expensive, time-consuming, challenging to tune, and lack of reproducibility across experiments and datasets (Han, 2016). As a result, researchers have long been searching for automatic metrics which are simple, generalizable, and which reflect human judgment (Papineni et al., 2002).

There are several sources of complexity in evaluating NLG output – the largest of which is there is often disagreement among humans about performance depending on the task (e.g. assigning a score to a generated paraphrase) (Rus et al., 2011). Most aspects of language do not decompose nicely into linear metrics and are task dependent (Novikova et al., 2017).

Automatic evaluation is a problem common to many areas in NLP which results in approaches being proposed from different domains which can also be applied to evaluating NLG. Machine translation, summarization, and image captioning are the most active areas in proposing automatic metrics which can be applied to NLG (Gatt et al., 2018).

There are broadly three categories of automatic metrics:

Word / N-gram Overlap

The core assumption embedded within word overlap metrics is that comparing sentences at the word level is sufficient to determine similarity. These metrics are by far the most widely used resulting from their ease of interpretability, implementation, and for historical benchmarking purposes (Gkatzia et al., 2015). One of the main downsides of this approach is it requires a corpus with ground truth labels which can be challenging to obtain. Another fundamental problem with this approach is it will have poor performance where valid generated sentences can deviate significantly from the ground truth (e.g. paraphrase generation). Word overlap metrics do not tend to directly consider grammatical structure which can have a significant impact on human judgment scores (Fomicheva, 2016). Using n-gram overlap instead of single words allows the metrics to capture a form of local grammatical structure (assuming the ground truth is grammatical) although this approach is very limited and does not consider sentence level structure or grammatical n-grams which are not contained in the ground truth.

The most common automatic metrics which are often applied to NLG problems are as follows:

- **BLEU (precision):** Normalized n-gram precision where the number of words in the generated text which appear in the reference sentences are adjusted for generated sentence length (Papineni et al., 2002).
- **NIST:** Similar to BLEU with a greater emphasis on less frequent n-grams and an adjusted penalty for sentence length (Doddington, 2002).
- **ROUGE (recall):** Similar in principle to BLEU although it measures the number of words in reference which appear in the generated sentence adjusted for length (Lin, 2004).
- **METEOR**: An alignment-based MT metric aimed to improve on BLEU by using a recall focused harmonic mean (F-10 measure), applying a stemmer, and also matching synonyms through WordNet (Lavie & Agarwal, 2007).

- **TER**: The minimum number of edits needed to change a hypothesis so that it exactly matches one of the references, normalized by the average length of the references. There are multiple variants of this approach including TERP and TERPA (Snover et al., 2006).
- **WMD:** The minimum amount of "distance" that the embedded words of one sentence need to "travel" to reach the embedded words of another sentence (Kusner et al., 2015).
- **CIDEr:** Applies term frequency-inverse document frequency (TF-IDF) weights to n-grams (stemmed) in the candidate and reference sentences, which are then compared by summing their cosine similarity across n-grams (Vedantam et al., 2015).

There have been many papers which highlight the poor performance of word level automatic metrics (Cui et al., 2018; Elliot & Keller, 2014; Callison-Burch et al., 2008; Novikova et al., 2017; Anderson et al., 2016; Liu et al., 2017; Vedantam et al., 2015) although for the reasons mentioned previously, evaluation using these metrics remains standard.

Semantic / Sentence Similarity

Given the limitations in word-level metrics we can instead consider the relationship between two sentences. This can help overcome the dependence on individual words by considering the relationship between words and the global sentence score including word context. There are two main approaches to evaluating sentence similarity:

Sentence Encoding Models

Similar to the success in capturing word-level semantics in word embeddings (Mikolov et al., 2013), sentence encoding models seek to embed the semantic structure into a single vector which can then be compared against another sentence embedding vector using a distance metric such cosine similarity. There are many models which aim to achieve sentence embeddings with some of the more popular approaches including:

- **Pooling over word embeddings** (Shen et al., 2018): word embedding models such as Word2Vec and GloVe revolutionized NLP demonstrating their ability to represent words in an embedding space capturing some characteristics of their semantics (Mikolov et al., 2013; Pennington et al., 2014). This largely solved the problem of sparsity, which occurred from prior bag of words approaches and led to strong performance in downstream tasks. As a result of the success of this approach many papers sought to extend this to the sentence and document level (Kiros et al., 2015; Iyyer et al., 2015; Le & Mikolov, 2014; Kalchbrenner et al., 2014; Tai et al., 2015; Arora et al., 2016; Socher et al., 2011).

- **InferSent** (Conneau et al., 2017): a specific example of sentence embedding approach, InferSent proposes a simple Bi-LSTM model with max-pooling that is pretrained on natural language inference tasks. While this approach is no longer state of the art, its ease of use and reasonable performance on transfer learning tasks warrants its consideration.
- **BERT** (Devlin et al., 2018): following the success of pretraining embeddings on downstream tasks, further research was done into contextual embeddings (Peters et al., 2018). BERT performs pretraining through training a large multilayer transformer model (Vaswani et al., 2017) on a masked bidirectional language modelling task on a large corpus. BERT achieved state of the art performance on eleven different NLP tasks and shows a reasonable understanding of linguistic structure and semantics (Devlin et al., 2018; Lin et al., 2019). While many similar models have subsequently come out such as XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019), and SpanBERT (Joshi et al., 2019) which achieve improved performance on many NLP benchmarks we treat these as variations on a theme and therefore consider BERT as a proxy for their performance.

While sentence encoding models are intuitive and work well in some tasks such as sentence classification, capturing the interactions between sentences is particularly important in paraphrase identification and determining semantic similarity (Lan & Xu, 2018).

Sentence Pair Interaction Models

Rather than embed the entire semantic structure of a sentence in a vector and compare the vectors as in the sentence encoding approach, sentence pair interaction modelling uses word alignment mechanisms and then models the inter-sentence interactions. There exist many models which seek to model inter-sentence interactions to achieve improved performance (Lan & Xu, 2018) although the main architectures we consider are as follows:

- **DecAtt** (Parikh et al., 2016): One of the earliest models using attention-based alignment for sentence pair modelling with a magnitude fewer parameters than other similar models. DecAtt computes the word pair interactions through a soft alignment which feeds the aligned phrases into another feedforward network which are then aggregated and concatenated for classification (Lan & Xu, 2018).

- **PWIM** (He & Lin, 2016): each word vector is encoded using LSTMs then every word pair across each sentence has its cosine similarity calculated and a hard attention is applied to the interaction similarities. A CNN is then applied to extract the features for classification. Although this model achieves strong performance, it has significantly longer computation time than other approaches as it must calculate similarities between all word pairs and then train a CNN on the interaction layers.
- **ESIM** (Chen et al., 2017): an improvement to DecAtt which uses Bi-LSTM to create bidirectional embeddings and average or max pooling instead of summation before classification.

Semantic Similarity Metrics

The SPICE metric (Anderson et al., 2016) seeks to measure the semantic overlap between two sentences through composing graphs between the objects and their relations and shows better correlation with human judgment than other image caption metrics. SPICE is specifically designed to evaluate generated image captions and leverages the fairly well-defined relational structure and importance of the correctly defined entities. SPICE has several drawbacks: it is not readily generalizable to other NLP problems, it is computationally expensive, and it ignores syntactic quality (Liu et al., 2017). SPIDEr was introduced as an extension to SPICE which is a linear combination of SPICE and CIDER that outperforms SPICE and other metrics (Liu et al., 2017).

Fluency / Grammar Score

While adequacy and fluency have long been considered the two main factors on which to evaluate generated text versus a reference (Toury et al., 2012; Stent et al., 2005; Gatt et al., 2018), a far greater amount of research has been dedicated to determining the adequacy score instead of fluency (Fomicheva et al., 2016). This is likely due to the challenges inherent in capturing what is considered a fluent sentence (Stent et al. 2005). Some approaches including GLEU (Mutton et al., 2007), SLOR (Kann et al., 2018), and Grammar Based Metrics (Napoles et al., 2016) have been introduced to measure what is considered a fluent sentence although they are not widely used in NLG. In research conducted by Martindale & Carpuat (2018), users responded strongly negatively to translations which were not fluent although were less concerned with adequacy in considering the trust of a system.

Despite the importance of fluency, current neural approaches do not explicitly model fluency as an objective and rather assume that MLE training objectives will capture general syntax rules (Linzen et al., 2016). While this works well in most cases, when the objective is changed from maximizing MLE to a reinforcement learning objective of maximizing an evaluation metric (such as BLEU) the model loses fluency (Liu et al., 2017). This results in the requirement of either explicitly evaluating fluency in the reward function or moving towards an adversarial approach where fluency is captured implicitly.

Application to ParaPhrasee

Understanding the existing approaches to automatic evaluation in NLP was integral to determining which approaches to try as reward functions for the RL model. We used BLEU, ROUGE, CIDEr, sentence-encoding similarity models, fluency models, and combinations of these metrics as a result of reviewing research into automatic evaluation approaches. We also used the ESIM model as our base model for the adversarial approach based on an evaluation of the different paraphrase identification models (Lan & Xu, 2018).

2.3 Deep Reinforcement Learning (DRL)

Reinforcement Learning studies a class of approaches to find actions which maximize the total reward an agent receives as it interacts with its environment (Sutton and Barto, 1998). Historically, reinforcement learning was constrained to problems with small state and action spaces given computational constraints (Arulkumaran et al., 2017). While applying neural networks as function approximators to games with large state spaces dates back to 1995 with TD-Gammon for backgammon (Tesauro, 1995), the renaissance of deep learning approaches has led to a significant amount of success in applying reinforcement learning to more complex environments.

RL agents have now been more successful than human players across many complex games including Go (Silver et al., 2016; Silver et al., 2017), Atari (Mnih et al., 2013), and StarCraft II (DeepMind 2019). One challenge with current reinforcement learning approaches in complex environments is they take a long time to train and do not tend to generalize well (Taylor et al., 2009; Parisotto et al., 2015). These problems can be addressed through the application of transfer learning which focuses on “transferring knowledge learned from different domains, possibly with different feature spaces and/or different data distributions” (Taylor et al., 2009; Pan and Yang, 2010; Weiss et al., 2016; Li et al., 2018).

Deep-Q learning is now one of the most well-known algorithms largely due to its success across the Atari suite of games (Mnih et al., 2013). A substantial amount of research has been done to extend and improve Deep-Q Learning approaches (Hessel et al., 2018), although extending value-based approaches to large action spaces is still an open research question (Zahavy et al., 2018). The more common approach in large action spaces is to use an Actor-Critic architecture in which the actor is a policy-based network and the critic is a value-based network (Silver et al., 2016; Silver et al., 2017). Monte Carlo Tree Search (MCTS) is also frequently applied as a planning algorithm in cases where models can be constructed (Liu et al., 2017; Silver et al., 2016). One of the downsides of MCTS is it requires a substantial amount of computation and memory (James et al., 2017).

Reinforcement Learning Approaches to NLG

Given the sequential structure of language, its dependency on prior context, and actions resulting in different future states, another approach to the language generation problem has been as structuring it as a planning problem (Lemon, 2008; Rieser & Lemon, 2009). This is particularly prominent in NLG for dialogue systems given the dynamics are even more sensitive to selected actions (Li et al., 2016). RL approaches to NLG have not been as widely explored as other neural approaches given the large state and action space, the challenges in defining a reward function, and the significant amount of computation required.

Success in applying adversarial generation approaches to image generation by Goodfellow et al. (2014) has led to a resurgence in the exploration of adversarial approaches to generation (Wang & Ward, 2019). However, the GAN framework cannot be applied directly to text generation as the gradients cannot be propagated through the network as the loss is non-differentiable in text generation. This has led to applying reinforcement learning architectures that are designed to follow a similar principle of adversarial generation (Yu et al., 2017; Che et al., 2017, Lin et al., 2017). Using reinforcement learning rather than maximum likelihood estimation for NLG has two significant advantages: the already discussed ability to optimize for a non-differentiable loss function, and overcoming exposure bias where the model has “only been exposed to the training data distribution instead of its own predictions” (Ranzato et al., 2015).

Application to ParaPhrasee

As the main contribution in this paper is understanding the impact of fine-tuning using different reward functions on paraphrase generation using reinforcement learning, it is important to select a good RL architecture. Reviewing the existing literature on RL approaches informed our selection of actor-critic policy-based architecture. It also highlighted the potential benefits of using model-based planning algorithms such as Monte Carlo Tree Search. Research into applying RL to complex state-action spaces led to the idea of pretraining using supervised learning being required to make the project feasible.

2.4 Paraphrase Generation

A paraphrase is “an alternative surface form in the same language expressing the same semantic content as the original form” (Madnani & Dorr, 2010) given the generated sentence is fluent. There have been many approaches to paraphrase generation and identification given it is an important subproblem in many NLP applications including information retrieval, chatbots, translation, and text summarization (Prakash et al., 2016). There are broadly three levels at which paraphrases can be considered:

- Word level (lexical): words with similar meanings (e.g. chair vs. seat)
- Phrasal level: fragments of words which contain similar meaning (e.g. he walked over to vs. he approached)
- Sentence level (sentential): complete sentences which convey similar meaning (e.g. Elephants like Dumbo like peanuts and bananas, and they can consume 20kg of food a day vs. an Elephants can eat up to 20kg of peanuts and bananas in a day).

We focus on sentential paraphrases as although they present the greatest technical difficulty, we feel they best capture the core objective of paraphrasing.

Traditional Approaches

Phrasal and sentential paraphrase generation was traditionally accomplished using hand-written rules (Fujita et al., 2008), formal grammars (McKeown, 1980; Dras, 1999; Gardent, et al., 2004;

Gardent and Kow, 2005), and machine translation approaches using monolingual and bilingual parallel corpora (Quirk et al., 2004; Bannard et al., 2005). These approaches are limited in their performance and are also time consuming to implement where manual rules are required (Prakash et al., 2016).

Neural Approaches

Following the successful application of neural approaches to machine translation (Sutskever et al., 2014) and other NLP problems, applying neural networks to the paraphrase generation task seemed to be a natural extension as it too can be framed as a sequence-to-sequence problem at the sentential and phrasal level. Kolesnyk et al. (2016) applied neural networks to generating an entailed sentence from a source sentence although generating paraphrases (bi-directional entailment) was not implemented until Prakash et al. later in 2016. Prakash et al., achieved state-of-the-art performance and demonstrated that their Residual LSTM model achieved the best performance across most of the metrics (BLEU, METEOR, Emb Greedy, and TER) and datasets (PPDB, WikiAnswers, and MSCOCO) that were evaluated.

Cao et al. (2017) use another vocabulary to limit word candidates in the generator model. Gupta et al. (2018) extend prior work in applying Variational Autoencoders (VAEs) to NLG (Hu et al., 2017) to the problem of paraphrase generation and are able to further improve the state of the art on the MS-COCO dataset for BLEU and METEOR without substantial hyperparameter tuning. Liu et al.'s (2018) use of policy-based reinforcement learning to optimize a generated image caption given an image is also related to our proposed approach to paraphrase generation although they do not apply an adversarial model as a reward function. Li et al.'s (2018) adversarial reinforcement learning approach is the current state-of-the-art in paraphrase generation and is the most related to our approach of the prior work.

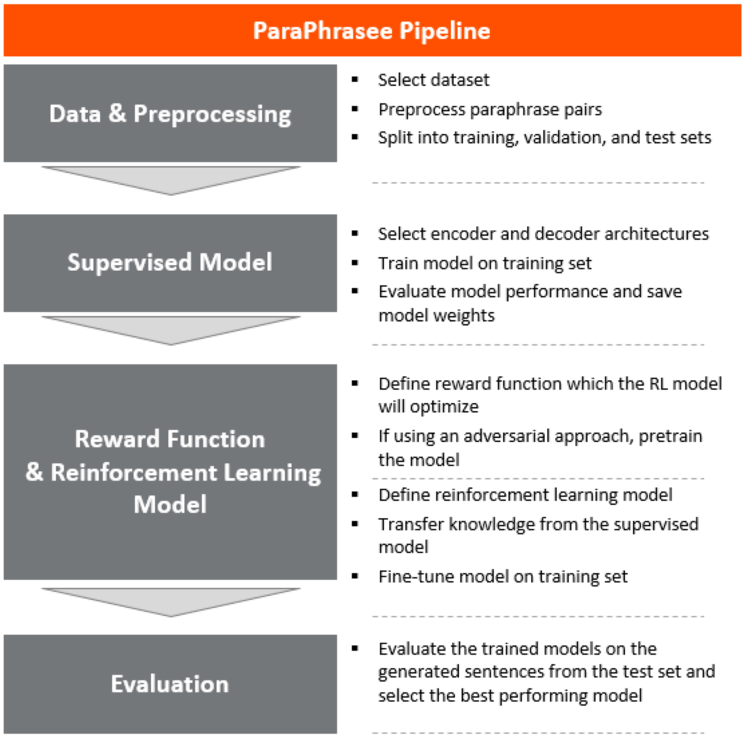
Li et al. (2018) address the challenges experienced in other paraphrase generation approaches including: exposure bias (Ranzato et al, 2015), the inability to optimize a non-differentiable function, and most importantly the lack of a meaningful evaluation measure. They propose the use of both a generator model (seq-to-seq supervised model) and discriminator (deep-matching model) which is first trained on a supervised learning objective and then fine-tuning on a reinforcement learning objective in which the reward is given by the discriminator. The evaluator is trained using both supervised learning and inverse reinforcement learning and achieves state-of-the-art results using different discriminators depending on the dataset. The elegance of this approach is it does not require a reward function which can be challenging to define.

Application to ParaPhrasee

As the main research question addresses paraphrase generation, it is important to understand what the strengths and limitations are of existing techniques as well as considering the history of paraphrase generation. Reviewing the traditional approaches highlights the inadequacy of generating handwritten rules leading to a focus on neural approaches. Ranzato et al. (2015) was incredibly useful in formulating the NLP generation problem as a RL problem. Li et al. (2018), was used as the inspiration for using an adversarial approach rather than a handcrafted composite reward function. Examining the other neural approaches such as Prakash et al. (2016) supported the concept of using an RNN architecture for paraphrase generation. Finally, Liu et al. (2017) showed that a model can be fine-tuned using RL to improve performance on MS-COCO for an arbitrary reward function.

3 Methods

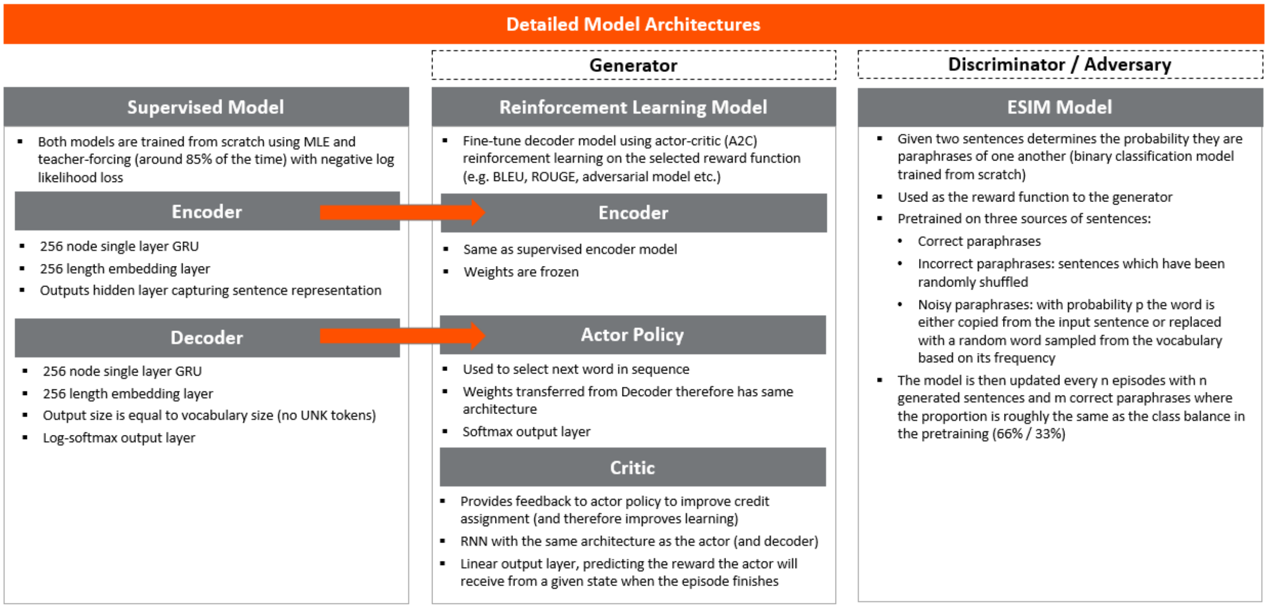
3.1 Pipeline Overview



ParaPhrasee consists of multiple steps in order to achieve high quality paraphrases using reinforcement learning in a computationally efficient way. A brief summary of the high-level pipeline is shown to the right which each section will expand upon.

A more detailed model architecture is provided below which outlines the base ParaPhrasee architecture including the adversarial model used.

Given the complexity in implementing reinforcement learning algorithms, two toy RL environments were developed (CartPole and FrozenLake) to test RL algorithms and find a repeatable strategy for transferring knowledge from the supervised model. These environments have a very different pipeline to the ParaPhrasee model and are covered in their respective sections within the reinforcement learning model section 3.5.



MCTS is applied to final decoder model to further improve performance

3.2 *Overview of Code Modules*

In order to answer the research questions, lots of coding was required which tied together ideas from different fields. Given the scale of the problem we leveraged the work of others where possible. While the Phrasee team was extremely helpful in discussing higher level architectural decisions, all the coding contribution was performed by me.

The project is structured into modules each of which with a short description below and most modules relying on code from other modules to eliminate redundant code. The full code for key modules is contained in Appendix D. Attribution of code is challenging given the large number of modules therefore I have estimated my contribution in terms of original lines of code module by module.

Overview of Code Modules

#	Module Name	Summary	Importance	# of Lines	% Original	Source
1	BFS_agent	Used to solve FrozenLake using breadth-first search to generate supervised training data	🟩	118	90%	Code snippets from StackOverflow
2	CartPole_solved	Used to solve CartPole using RL to generate supervised training data	🟩	319	25%	Forked from: https://github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py with minor changes to run and save the data Made changes to transferring model to supervised model and saving data
3	cart_pole_env	Defines environment dynamics for CartPole problem	🟩	103	10%	Forked from: https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py Removed code pertaining to rendering
4	config	Holds all global variables, Cuda settings, and file paths to facilitate changing between machines (e.g. server and local computer)	🟩🟩	88	100%	-
5	create_ESIM_data	Creates training set to train ESIM model	🟩	100	100%	-
6	data	Imports raw data from various sources, preprocesses, creates train/test sets and vocab index Also contains functions for saving and loading	🟩🟩🟩	546	90%	Non-original code is saving and loading snippets from Stack Overflow
7	demo	A short demo for a presentation at Phrasee highlighting generation using supervised learning and RL	🟩	62	100%	-
8	encoder_models	Defines classes for each encoder: GloVe, BERT, InferSent, Vanilla and GPT language model along with related code	🟩🟩🟩	247	*80% **	* While the code in this module is largely original, it is mostly a wrapper for the underlying encoder packages BERT is implemented using the transformers library (https://github.com/huggingface/transformers) and the fine-tuned version (https://pypi.org/project/sentence-transformers/) Similarly, we used FAIR's InferSent implementation (https://github.com/facebookresearch/InferSent) with minor modifications We also used Magnitude's GloVe implementation (https://github.com/plasticityai/magnitude)
9	evaluate_model_results	Generates sentences from trained models and evaluates performance on selected evaluation metric	🟩🟩	285	100%	-
10	frozen_lake_env	Defines environment dynamics for FrozenLake problem	🟩🟩	177	10%	Forked from: https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py Removed code pertaining to rendering and simplified step function
11	InferSentModels	InferSent model PyTorch implementation	🟩	818	0%	Forked from: https://github.com/facebookresearch/InferSent

Overview of Code Modules (Continued)

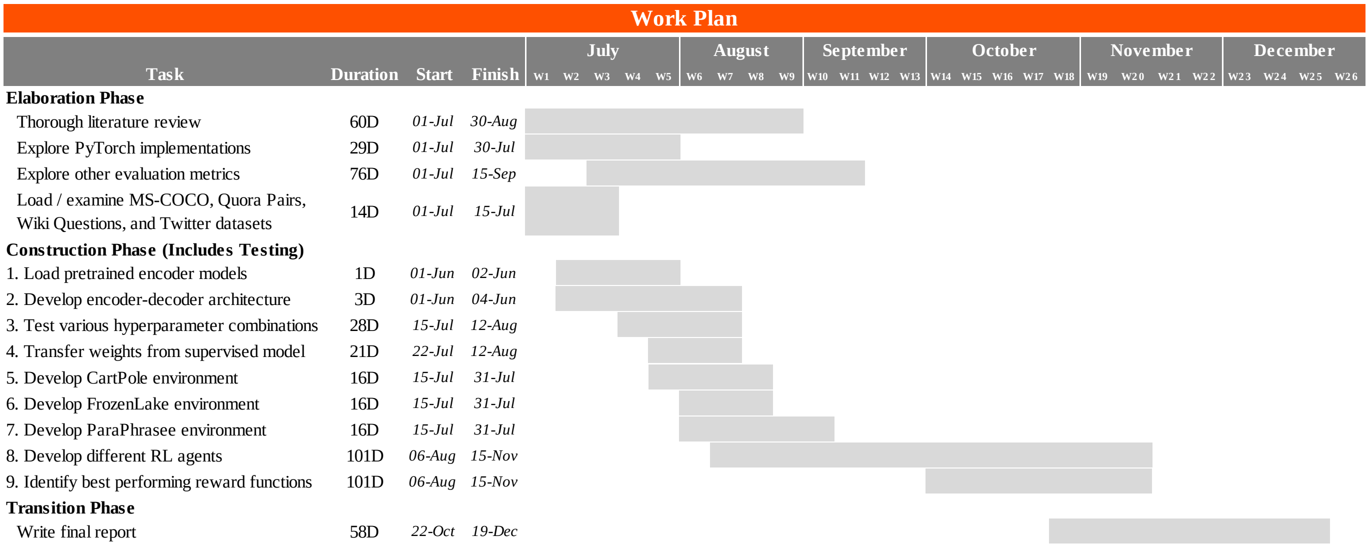
#	Module Name	Summary	Importance	# of Lines	% Original	Source
12	main	Used to get outputs for the report and run ad-hoc experiments	🟩	236	100%	-
13	MCTS	Monte Carlo Tree Search implementation for both FrozenLake and ParaPhrasee environments	🟩🟩🟩	216	50%	Forked from: https://github.com/brilee/python_uct/blob/master/numpy_impl.py Significant modifications were required to be able to run our environments
14	model_evaluation	Contains wrappers for different evaluation functions to be used primarily as reward functions for RL model	🟩🟩🟩	386	*90% **	* While the code in this module is largely original, it is mostly a wrapper for the underlying functions The main reward functions (BLEU, ROUGE, METEOR, CIDEr) are a modified version of the Coco API's evaluation tool: https://github.com/tylin/coco-caption. Similarly, we used other libraries for the sentence encodings as discussed in the encoder_models module
15	paraphrasee_env	Defines environment dynamics for paraphrase generation task as RL problem	🟩🟩🟩	128	100%	-
16	reddit_comment_data	Loads raw reddit data and preprocesses	🟩	89	100%	-
17	reddit_model	Trains simple logistic regression on sentence embeddings on Reddit comment data	🟩	90	100%	-
18	RL_model	Defines and trains RL models for ParaPhrasee environment	🟩🟩🟩	690	80%	PyTorch starter code: https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py https://github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py
19	supervised_metric_dist	Code to get the reward distribution from the supervised models to scale the different metrics when combining	🟩	98	100%	-
20	supervised_model	Defines, trains, and evaluates the defined supervised model with MLE. Includes modifications for teacher forcing and attention.	🟩🟩🟩	907	60%	Beam decoding: https://github.com/budzianowski/PyTorch-Beam-Search-Decoding/blob/master/decode_beam.py PyTorch starter code: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
21	toy_RL_pipeline	Defines and trains RL models for either CartPole or FrozenLake environments	🟩🟩🟩	883	70%	PyTorch starter code: https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py https://github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py
22	train_ESIM	Defines and trains an ESIM model	🟩🟩🟩	356	*80% **	* While the code in this module is largely original, we forked Lan & Wu's excellent ESIM implementation which underlies this although we updated it for Python 3 and PyTorch 1.0+ https://github.com/lanwuwei/SPM_toolkit/tree/master/ESIM
23	tree_space_env	Defines environment dynamics for TreeSpace problem	🟩	200	80%	Forked underlying tree building logic from MCTS implementation (https://github.com/noahwaterfieldprice/alphago/blob/master/alphago/mcts_tree.py)
24	utils	Contains utilities used throughout the code such as converting sentences to tensors, layer normalization, and plotting	🟩🟩	246	80%	Non-original code is snippets from Stack Overflow and from supervised learning PyTorch starter code

For jobs which were more computationally intensive we used City, University of London’s server which required developing shell scripts that are outlined below. Using the server added a layer of complexity in terms of ensuring file paths were consistent, using Git to ensure code version consistency, and learning how to use the resource manager Slurm in addition to bash commands. Using the server allowed us to run more involved experiments and run experiments concurrently.

Slurm Shell Scripts

#	Script Name	Summary
1	CartPole_single_model	Trains a single model
2	FrozenLake_single_model	
3	ParaPhrasee_single_model	
4	ESIM_single_model	
5	ParaPhrasee_pretrain	Pretrains critic for ParaPhrasee
6	create_ESIM_data	Creates training data for ESIM
7	train_CartPole_RL_bash_script	Trains multiple models in sequence
8	train_FrozenLake_RL_bash_script	
9	train_supervised_models	
10	train_ParaPhrasee_RL_bash_script	
11	generate_RL_model_preds	Generates predicted sentences given trained model
12	generate_supervised_model_preds	
13	eval_FrozenLake_RL_bash_script	Evaluates performance of model's predictions
14	eval_RL_models	
15	eval_supervised_models	
16	MCTS_ParaPhrasee_RL_bash_script	

An illustrative work plan is below which details the approximate allocation of time spent on the various tasks throughout the project.


Work Plan

Task	Duration	Start	Finish
Elaboration Phase			
Thorough literature review	60D	*01-Jul*	*30-Aug*
Explore PyTorch implementations	29D	*01-Jul*	*30-Jul*
Explore other evaluation metrics	76D	*01-Jul*	*15-Sep*
Load / examine MS-COCO, Quora Pairs, Wiki Questions, and Twitter datasets	14D	*01-Jul*	*15-Jul*
Construction Phase (Includes Testing)			
1. Load pretrained encoder models	1D	*01-Jun*	*02-Jun*
2. Develop encoder-decoder architecture	3D	*01-Jun*	*04-Jun*
3. Test various hyperparameter combinations	28D	*15-Jul*	*12-Aug*
4. Transfer weights from supervised model	21D	*22-Jul*	*12-Aug*
5. Develop CartPole environment	16D	*15-Jul*	*31-Jul*
6. Develop FrozenLake environment	16D	*15-Jul*	*31-Jul*
7. Develop ParaPhrasee environment	16D	*15-Jul*	*31-Jul*
8. Develop different RL agents	101D	*06-Aug*	*15-Nov*
9. Identify best performing reward functions	101D	*06-Aug*	*15-Nov*
Transition Phase			
Write final report	58D	*22-Oct*	*19-Dec*

3.3 Data

Given defining the characteristics of a paraphrase is challenging (Bhagat & Hovy, 2013), numerous datasets have been used which capture different levels of paraphrase quality. We focus on sentential level paraphrase generation and therefore only consider datasets which can be used to achieve this objective.

- **Microsoft Paraphrase Corpus** (Dolan & Brockett, 2005): One of the first widely used paraphrase benchmarks which contains 5,801 sentence pairs that are hand-labeled whether a pair is considered a paraphrase or not. The corpus was created through defining an algorithm to automatically determine whether two extracted probable candidate sentences are in fact paraphrases. Given its relatively small size and lack of use as a benchmark we did not use this dataset in our experiments.

Sentence 1	Sentence 2	Score
Amrozi accused his brother, whom he called "the witness", of deliberately distorting his evidence.	Referring to him as only "the witness", Amrozi accused his brother of deliberately distorting his evidence.	1

- **Semantic Text Similarity Competitions** (Agirre et al., 2017): SemEval is a series of tasks as part of the Association for Computational Linguistics. The dataset consists of a selection of the English datasets used in semantic text similarity tasks between 2012-2017. There are three sources of sentence pairs: news headlines (4,299), image captions (3,250), and messages from forums (1,079) resulting in 8,628 sentence pairs total. The label is a rating from 0-5 of how semantically similar the sentences are. There were multiple issues with using this dataset: the relatively low size, the lack of use as a benchmark, a better image caption dataset is contained in MS-COCO, and STS values of ~5 tended to be almost verbatim sentence pairs (further reducing the size of the dataset).

#	Sentence 1	Sentence 2	Score
1	The man is riding a horse.	A man is riding on a horse.	5
2	No I show you are willfully ignorant.	LOL	4
3	China on high alert for typhoon Kalmaegi	China issues yellow alert for typhoon Kalmaegi	4

- **Wiki Answers** (Fader et al., 2013): Wiki Answers is a question asking / answering platform where users can post a question and other users post the reply. Given the questions are user generated, it is quite common that it is a duplicate of another existing question and therefore the option exists to tag it as a duplicate. Fader et al. (2013) consider duplicate questions as paraphrases in order to improve their question answering application. While this dataset is very large (~18 million pairs), there has been no attempt to correct for grammar and what is considered a duplicate question is very dubious. The pairs have also been heavily processed linguistically (lower cased, lemmatized) and this is not used as a benchmark in two out of three papers therefore we did not use this dataset in our experiments.

#	Sentence 1	Sentence 2	Score
1	a young lizard?	what be young lizard call?	1
2	have demi lovato kiss anyone?	what be demus lovato background?	1
3	5 cubic foot equal how much?	what be 1 cubic foot?	1

- **Twitter Shared Links** (Lan et al., 2017): In order to develop a corpus at scale which specifically captures paraphrase quality, Lan et al. (2017) rely on tweets which link to a common URL in a Twitter post. They then provide each Amazon Mechanical Turk worker with an input sentence and 10 candidate sentences and asked them to select the sentences with the same meaning. Each pair was then considered a paraphrase pair (or non-paraphrase pair) based on majority vote of the workers. The score is the proportion of raters which agreed. The pairs were then checked for inter-rater agreement and agreement with an expert for a subset of pairs. Lan et al. then train a paraphrase detection model to create a large database of over 2.8 million tweets that are likely to be paraphrases with over with 70% precision. While this database is much more useful for generating paraphrases than the datasets above, the use of language on Twitter is very different from standard language and many of the "perfect paraphrase" pairs (6/6 score) are identical or nearly identical sentences, therefore we did not use this dataset in our experiments.

#	Sentence 1	Sentence 2	Score
1	The '#Impossible' #Veggie Burger Answer to Big Mac.	The Impossible Burger is the veggie burger that "bleeds". So how does it taste?	6/6
2	Shame on you rips #Dems who plan to skip inauguration.	Shame on You a hero cause he was beaten when he was an activist? Hes still an activist isn't he?	4/6
3	This newly discovered species of moth has been named after Donald Trump . Can you guess why ?	New #moth named in honor of Donald Trump @CNN	6/6

- **Quora Duplicate Questions** (Kaggle, 2017): Quora is a website similar to Wiki Answers in which users ask questions and other users answer them. Quora released a dataset consisting of over 400k sentence pairs containing potential duplicates (~155k actual duplicate questions) as part of a Kaggle data science challenge. As the text quality is much higher than Wiki Answers, both Gupta et al. (2018) and Li et al. (2018) use the duplicate questions portion as supervised data for training and evaluating their models.

#	Sentence 1	Sentence 2	Score
1	Do you believe there is life after death?	Is it true that there is life after death?	1
2	Fitness: What can I do to reduce my bulky tummy?	How do I reduce tummy?	1
3	How do I improve at drawing?	How do you improve your drawing?	1

- **Microsoft Common Objects in Context (MS-COCO)** (Lin et al., 2015): As a result of the success of deep learning in many computer vision applications following the application of CNNs to object recognition (Krizhevsky et al., 2012), the research community increased its focus on scene understanding. The dataset contains 120k images of common scenes with 5 image captions per image provided by 5 different human annotators. The image captions tend to have a fairly defined structure of A <NOUN> is <DOING SOMETHING> in <LOCATION DESCRIPTION> as seen in some of the examples below. Both Prakash et al. (2016) and Gupta et al. (2018) use the image captions for the same image as supervised data for training and evaluating their models.

  As the descriptions describe the same image, they tend to have high semantic similarity and represent interesting paraphrases as the use of language tends to differ. However, given the image captions are provided by different people they tend to have slight variations in detail provided in the scene therefore the generated sentences tend to hallucinate details when using supervised learning. In spite of this, given its size, cleanliness, use as a benchmark for two notable paraphrase generation papers, we use MS-COCO as the primary benchmark in our analysis.

#	Sentence 1	Sentence 2	Score
1	A black Honda motorcycle parked in front of a garage.	A Honda motorcycle parked in a grass driveway.	1

2	Three teddy bears sit in a sled in fake snow.	A set of plush toy teddy bears sitting in a sled.	1
3	A cat eating a bird it has caught.	A white cat caught a bird outside on a patio.	1

Dataset Preprocessing

For the reasons discussed above, the MS-COCO image captioning dataset was selected as the basis of our analysis. We consider a paraphrase to be any two captions describing the same source image therefore the first task is to extract all the image captions and convert the dataset into pairs of paraphrase sentences. Next, we remove sentences above a certain length threshold to ensure comparable input and output sentence lengths. The maximum token length selected was 12 tokens.

We then preprocess the sentences by removing numbers (e.g. 0-9 although leaving the word one, two, etc.), lowering the case of the sentence, removing punctuation other than <.!?>, and adding a space before punctuation tokens in order to create punctuation embeddings.

Given the limited computational resources available and in order to improve the speed of the experimentation, we only consider a randomly selected sample of the full dataset for the experimentation. We shuffle the data and then randomly select 100k sentence pairs and split it into train (65%), validation (25%), and test sets (10%). As is the common best practice in machine learning, the training set is used for training the parameters of the model, the validation set is used for selecting model hyperparameters and early stopping, and the test set is used for providing an unbiased estimate of the performance of the final model in production. A vocabulary index is then created from the dataset for use in training the PyTorch models which maps each token to an index value. We do not assign out-of-vocabulary tokens which may be an area of future improvement as suggested in the future works section.

Deep Learning Frameworks and Python Packages

In developing the project, we selected PyTorch as our automatic differentiation framework for use in building the deep learning models. TensorFlow and its higher-level implementation, Keras, were also considered although the strong research community, particularly for NLP and RL, which has developed around PyTorch made it the clear choice.

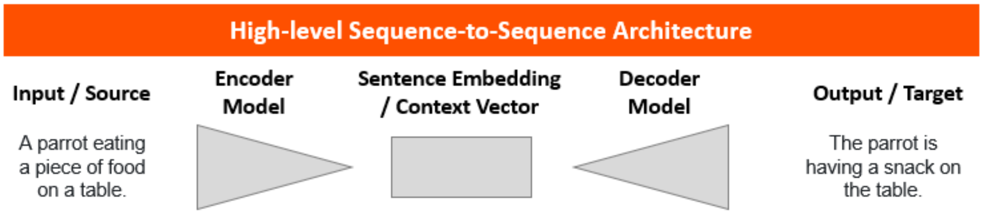
As discussed later, we relied on several key packages and implementations in achieving the results. We used HuggingFace's PyTorch-Transformers package for BERT, GPT, and GPT-2 implementations *(https://github.com/huggingface/transformers)*, a modified version of Lan & Xu's (2018) ESIM model implementation, and a modified version of Coco API's evaluation metrics *(https://github.com/tylin/coco-caption)* to ensure consistency in the evaluating the reported metrics with other prior work.

3.4 Supervised Model

Model Training Procedure and Architectures

While reinforcement learning has improved considerably in terms of sample efficiency, training times, and overall best practices, training RL models from scratch is still comparatively very slow and unstable (Arulkumaran et al., 2017). Supervised learning is far better understood than reinforcement learning with mathematical guarantees around convergence, even for newer deep learning approaches (Vidal et al., 2017). While one common solution to overcoming this instability and slow training time in RL models is to scale the computation up and out (Mnih et al., 2016; Ganin et al., 2018), given our computational constraints, we instead focus on approaches which leverage transferring knowledge obtained through supervised pre-training (Ranzato et al., 2015). Transferring knowledge from the supervised model dramatically decreases training time for the RL model at the cost of reduced exploration.

ParaPhrasee first starts with training a supervised model on the paraphrase generation problem framed as a sequence to sequence task (seq-to-seq) with a maximum-likelihood estimation (MLE) objective. As discussed in section 2.1, the dominant approach is to use an encoder network and decoder network (which serves as a conditional language model) (Dusek et al., 2018).



A vanilla RNN approach to supervised generation consists of an encoder model which aims to embed the information from the input sentence into a context vector which is used by the decoder model to predict each word given the context until the sentence is completed.

In order to train the paraphrase generation model using reinforcement learning, we only consider fine-tuning the decoder network and treat the encoder network as static. As a result, models which produce a sentence embedding vector, rather than attention-based models are preferred. Attention-based approaches, are also considered although the purpose of training the supervised model is to warm-start the RL model rather than achieve state-of-the-art performance directly.

Teacher-forcing is a common approach to training sequential models (such as RNNs) in which rather than use the models potentially incorrect predictions at the next time step, the model is fed the correct ground truth (Goodfellow et al., 2016). For example, if the correct reference sentence is “a cat is sitting in the chair.” and the model had thus far predicted “a cat runs” it is extremely unlikely the next predicted word is “sitting”. To remedy this, teacher forcing corrects the model’s error and instead asks the model to predict the next word given “a cat is” where predicting “sitting” is now far more likely.

The significant downside to teacher-forcing is when the model is asked to generate a new sentence it experiences exposure bias in which any prediction error compounds significantly as the state gets further from that which the model seen during training (Ranzato et al., 2015). The trade-off is then between training time / efficiency and resilience to prediction error during generation. To balance this trade-off, we set a probability threshold hyperparameter where the model will either use teacher-forcing or the predicted word. This threshold decays linearly over the training period between start and end values as the model learns more.

The models were trained using online learning and tested using different optimizers including stochastic gradient descent (SGD), Adam (Kingma & Ba, 2014), and starting with Adam and switching to SGD to balance speed and performance (Keskar & Socher, 2017).

The following supervised encoder-decoder model architectures were evaluated:

- **Vanilla encoder-decoder model:** The encoder-decoder framework introduced by Sutskever et al. (2014) was used as the base model. Each token in a sentence is assigned a randomly initialized word embedding which is adjusted throughout training. The sentence encoding also called context vector which is passed to the decoder network is the final hidden state of the GRU network after passing over each word vector. While the vanilla model showed very strong performance, we also considered other encoders to create sentence embeddings.

    - **Pooled word embeddings:** instead of using an RNN for the encoder network we can instead just max or mean pool the word embeddings to get a sentence embedding (Shen et al., 2018). For our experiments we only consider mean pooling as the results are comparable.
    - **InferSent encoder (Conneau et al., 2017):** as discussed in section 2.2, InferSent is a relatively simple Bi-LSTM model with max-pooling that is pretrained on natural language inference tasks.
    - **BERT encoder (Devlin et al., 2018):** as discussed in section 2.2, BERT is a large multilayer transformer model (Vaswani et al., 2017) on a masked bidirectional language modelling task which was pretrained on a large corpus.
- **Attention model:** In contrast with vanilla encoder-decoder networks, attention networks do not create a single encoding vector for the sentence and instead create a weighted sum of the input vectors at each decoding step which is dependant on the decoder's input context (Vaswani et al., 2017). This reduces the ability to make the problem modular with a trained encoder producing a sentence embedding. For simplicity we do not focus on fine-tuning attention models using RL although this is likely to improve performance and has been left for future work.

As discussed in section 2.2, automatic evaluation of NLG models is still an open research problem and therefore selecting a supervised model to warm-start the reinforcement learning model is not a trivial problem. We consider error on the test set, model complexity, and qualitative generation performance as factors in model selection.

Beam search is a commonly used approach to generating sentences using language model decoders by selecting the top n most probable words at each step and building out a graph of these words to get the most probable sentences (Sutskever et al., 2014). Beam search does not directly allow you to optimize for a specific metric although will improve generation coherence. The greater the depth of the beam search the more common the sentences are and therefore the less likely they are to be ungrammatical and ill-formed. However, the less likely the sentences are to contain rare words and descriptive features which results in potentially boring outputs.

Implementation and Project Information

The beginning supervised model code was from a PyTorch machine translation tutorial (*https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html*). This code served as the framework upon which the many improvements were built on. InferSent was used out-of-the-box from the open-source code after making minor tweaks in order to run it (*https://github.com/facebookresearch/InferSent*). BERT requires fine-tuning to produce higher quality sentence embeddings. Therefore we used the sentence-transformers package which fine-tunes BERT using a Siamese network structure to produce semantically useful sentence embeddings (*https://pypi.org/project/sentence-transformers/*). The beam search code was based on (*https://github.com/budzianowski/PyTorch-Beam-Search-Decoding/blob/master/decode_beam.py*) with minor changes. Layer norm was used for regularization to stabilize the performance of the GRU networks when using pretrained encoders and was based on a code snippet (*https://github.com/pytorch/pytorch/issues/1959*).

3.5 Reinforcement Learning Model

Model Training Procedure and Architecture

In order to efficiently fine-tune the paraphrase generation model using reinforcement learning, we only consider fine-tuning the decoder and treat the encoder as static. This means the encoder is either pretrained using supervised learning per the vanilla architecture discussed in 3.4 or it is from a pretrained sentence encoding model such as BERT or InferSent. The problem can then be stated as given a specified reward function how should the model update the weights in the decoder to maximize the expected long-term reward. This contrasts with maximizing the probability that the next predicted word is correct versus a ground truth (MLE objective) as it is instead using a planning approach to text generation and will result in generated paraphrases that better capture the reward function at the sentence level.

Reinforcement learning studies a class of approaches to find actions which maximize the total reward an agent receives as it interacts with its environment (Sutton & Barto, 1998). The two key components of a RL problem are the agent and the environment. The environment is composed of four key components: the state, the action space, the transition function, and the reward function.

- **The state:** the information the agent receives to select its action and should encapsulate "past sensations compactly, yet in such a way that all relevant information is retained" (Sutton & Barto, 1998). States which accomplish this are said to follow a Markov Decision Process (MDP) meaning "The future is independent of the past given the present" (Alonso & Mondragón, 2019).
- **The action space:** the available actions an agent can take in a given state.
- **The transition function:** the dynamics of the environment dictate if an agent is in a given state and takes an action what the resulting state the agent will be in. Transition functions can be either deterministic or stochastic which impacts the performance of the RL agents.
    - *Deterministic transition function:* given the agent is in a state and takes an action it will result in the same end state each time. For example, in chess selecting the action of moving the pawn from space D2 to D4 will always result in the pawn ending in D4.
    - *Stochastic transition function:* given the agent is in a state and takes an action it will result in a different state determined by a probability distribution that is defined by the state-action pair. For example, if you are on an icy and windy road and try to move a remote-controlled car forward, depending on the wind and ice, the car will end in a forward position with some probability and a left or right position with some probability.
- **The reward function:** the value an agent receives by moving to a new state by taking an action from its current state. As this is the only feedback the agent receives and is what the agent is trying to maximize, the reward function is arguably the most important component of defining a RL problem.

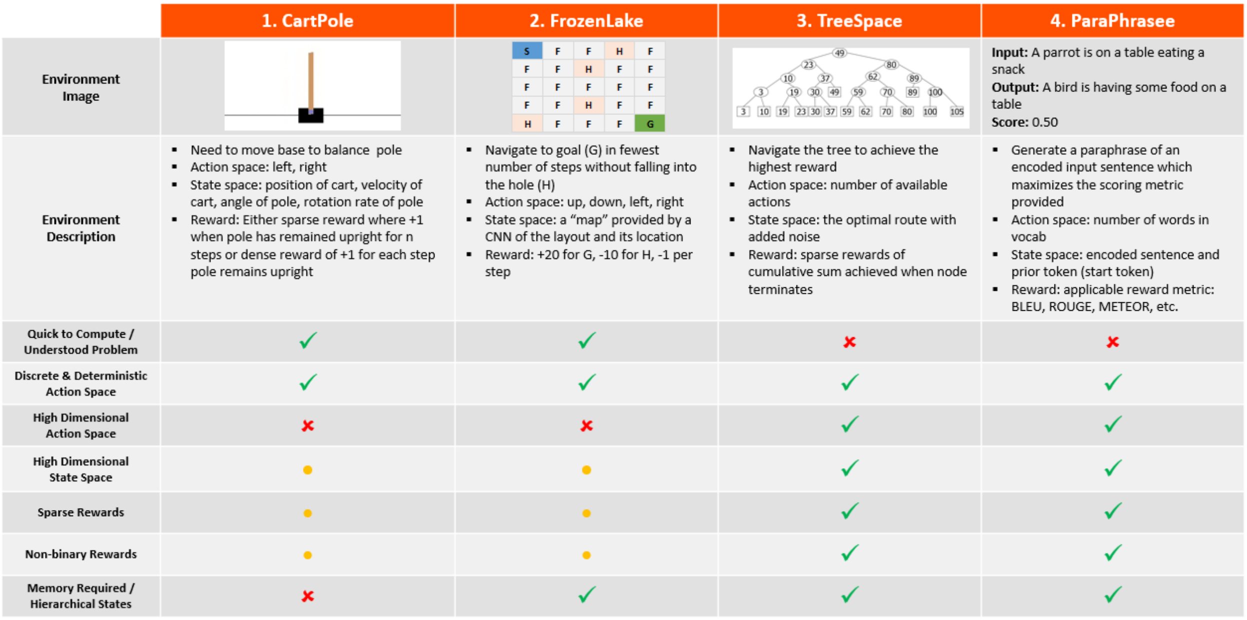
Solving paraphrase generation as a RL problem is very challenging as it has a very complex state and action space which are both massive. For ParaPhrasee the problem can be structured as follows:

- **The state:** the start state is the encoded sentence concatenated with the learned word embedding for the start token. The subsequent states consist of the prior hidden state from the actor model used in making its prior word prediction concatenated with the learned word embedding for the prior predicted token.

- **The action space:** the number of words in the vocabulary including start and end tokens.
- **The transition function:** for a specific episode it is deterministic, although it changes over time as the actor trains. This means that for a given state, if the model selects a specific action (word choice) then the next state will be the same representation were the experiment to be repeated. This does not mean that for a given input sentence and partially predicted sentence that the state will be the same throughout the training process. As eluded to, this is because the state representation will update as the model updates therefore the model is highly unstable.
- **The reward function:** the specified automated evaluation metric: BLEU, ROUGE, METEOR, etc. Once a predicted sentence has been completed, the metric evaluates its performance relative to the reference sentence.

As such, in developing algorithms which would be well suited to handling the ParaPhrasee environment, we proposed three intermediary environments for testing as sub-environments. A list of elements an environment must contain in order to be similar to the ParaPhrasee environment to reduce the list of potential environments was created.

Intermediate Environments Towards ParaPhrasee



	1. CartPole	2. FrozenLake	3. TreeSpace	4. ParaPhrasee
Environment Image		S F F H F F F H F F F F F F F F F H F F H F F F G		**Input:** A parrot is on a table eating a snack **Output:** A bird is having some food on a table **Score:** 0.50
Environment Description	▪ Need to move base to balance pole ▪ Action space: left, right ▪ State space: position of cart, velocity of cart, angle of pole, rotation rate of pole ▪ Reward: Either sparse reward where +1 when pole has remained upright for n steps or dense reward of +1 for each step pole remains upright	▪ Navigate to goal (G) in fewest number of steps without falling into the hole (H) ▪ Action space: up, down, left, right ▪ State space: a “map” provided by a CNN of the layout and its location ▪ Reward: +20 for G, -10 for H, -1 per step	▪ Navigate the tree to achieve the highest reward ▪ Action space: number of available actions ▪ State space: the optimal route with added noise ▪ Reward: sparse rewards of cumulative sum achieved when node terminates	▪ Generate a paraphrase of an encoded input sentence which maximizes the scoring metric provided ▪ Action space: number of words in vocab ▪ State space: encoded sentence and prior token (start token) ▪ Reward: applicable reward metric: BLEU, ROUGE, METEOR, etc.
Quick to Compute / Understood Problem	✓	✓	✗	✗
Discrete & Deterministic Action Space	✓	✓	✓	✓
High Dimensional Action Space	✗	✗	✓	✓
High Dimensional State Space	●	●	✓	✓
Sparse Rewards	●	●	✓	✓
Non-binary Rewards	●	●	✓	✓
Memory Required / Hierarchical States	✗	✓	✓	✓

Each environment is discussed in greater detail in its respective section below although a list of potential environments was considered and narrowed down to CartPole, FrozenLake, and

TreeSpace. CartPole and FrozenLake are two common benchmark environments for developing RL algorithms although they have been modified to be more similar with the ParaPhrasee task.

TreeSpace was developed although abandoned as the environment was too simple to be comparable to ParaPhrasee. To increase its complexity in a measured way proved to be a more challenging task than solving ParaPhrasee using the insights gained from CartPole and FrozenLake environments.

Reinforcement Learning Algorithms

The two main approaches to solving control problems in RL are value-based and policy-based algorithms. In value-based approaches such as Q-Learning and its variations, the model learns to estimate the value (expected return) associated with a state-action pair and therefore if it selects the maximum value and continues to select the maximum values for each state-action pair thereafter it will achieve the optimal policy (Alonso & Mondragón, 2019; Sutton & Barto, 2018). The challenge with this approach is it requires visiting each state-action pair many times (or very similar state-action pairs) in order to train a neural network which can accurately interpolate the state-action values.

While the difficulty in inference clearly depends on the complexity of the environment, this results in an intractable amount of computation for ParaPhrasee with an action space equal to the vocabulary (over 15,500 tokens for sampled environment) and a state space represented by an RNN neural network. Another problem with value-estimation approaches is they do not naturally lend themselves to stochastic policies as during generation the “best” action is to select the maximum value.

Policy-based approaches instead optimize the best policy directly through acting on-policy and updating the model parameters based on the observed rewards (Sutton & Barto, 2018). There are multiple benefits to this approach for the ParaPhrasee environment. The model does not need to learn any value functions which depending on the dynamics of the environment can be more challenging to learn than the optimal policy, it naturally results in a stochastic policy, and it more easily allows knowledge contained in the supervised learning model to be transferred. The most well known policy-gradient RL algorithm is REINFORCE, which completes an episode and uses gradient ascent to update the model parameters to increase the probability of the actions which resulted in a high score (Sutton et al., 2000). The main weakness of this approach is the high variance caused by the credit assignment problem which describes the difficulty in attributing the reward to specific actions. The solution to this is to remove a baseline value which is decorrelated with the action such as a constant value (Neubig, 2019). While this decreases variance in policy-based approaches, using a model to better approximate the improvement in score resulting from a specific action significantly improves performance (Sutton & Barto, 2018).

Approaches which estimate the state-value to improve credit assignment and reduce variance in policy-gradient methods are called Actor-Critic methods. These approaches either have a separate model which given a state estimates the corresponding value (expected return) or do multi-headed learning and have the same network approximate both. Actor-Critic approaches have been very successful in application (Silver et al., 2016; Silver et al., 2017, Liu et al., 2017) as they scale easily to large problems both in terms of complexity and computationally (Mnih et al., 2016). For this reason, we apply Actor-Critic as the main architecture in our experiments across environments.

Model-based RL methods "rely on planning as their primary component" for defining a policy (Sutton & Barto, 2018) and can be far more efficient in terms of learning and adapting to changes in the environment than model-free alternatives (Hasselt, 2018). Monte Carlo Tree Search (MCTS) is a model-based planning approach in which the agent simulates rollouts a specific state and subsequent actions in order to estimate the value of a state. It has been very successfully applied to games and many other complex problems in reinforcement learning (Browne et al., 2012; Silver et al., 2016; Silver et al., 2017, Liu et al., 2017).

One of the challenges with model-based approaches is if the model is deficient then the learned policy will not be optimal (Brunskill, 2019). As MCTS requires completing many rollouts and the performance improves with the number of rollouts it can result in a very significant increase in computation (Brunskill, 2019). In their landmark research applying MCTS to produce a world-leading Go agent, Silver et al. (2016) stated using a neural network to approximate the value as in Actor-Critic "approached the accuracy of Monte-Carlo rollouts using the RL policy network but using 15,000 times less computation". Given the significant trade-off in computation versus performance we only use MCTS as a form of beam-search when the models are already fully trained for ParaPhrasee.

Approaches to Efficient Knowledge Transfer

In order to improve computational efficiency in the performance of the RL agent we considered various strategies to transferring the knowledge from the supervised model to the reinforcement learning decoder model.

- **Weight initialization:** the simplest approach is to transfer the weights from the supervised model to the reinforcement learning model (Silver et al., 2016). However, this approach requires maintaining the same neural architecture between models and if the learning rate for the RL model's optimizer is too high then the model can quickly diverge and the performance can suffer dramatically (Montone et al., 2017).
- **Model switching:** the second approach is similar to the MIXER algorithm developed by Ranzato et al. (2015) and involves randomly switching between using the supervised model to make a prediction and the reinforcement learning model at each step. This approach requires maintaining two models while generating although the intuition is that it would cause the RL model not to wander far from the supervised predictions. The switch proportion would then be annealed based on a schedule. This approach was abandoned as it failed to show meaningful results as discussed in the Results section.
- **Policy distillation:** distilling the knowledge from a large deep neural network into a shallower network has been widely explored (Cheng et al., 2017; Hinton et al., 2015). Distillation in a RL context has been less explored although some prior work exists (Rusu et al., 2016; Parisotto et al., 2016; Schmitt et al., 2018). Schmitt et al.'s approach to "Kickstarting Deep Reinforcement Learning" involves adding an auxiliary loss function which measures the difference in prediction between the teacher (trained supervised model) and the RL model. We implemented the following algorithm which is closest in similarity to the work done by Schmitt et al (2018).
    - Train supervised encoder-decoder model using log-softmax output
    - Convert logits produced by decoder to probabilities using Temperature hyperparameter (where higher T represents softer distribution and T=1 is equal to softmax)
    - Transfer weights from supervised model to RL actor with same architecture
    - Train RL model using by adding auxiliary loss term

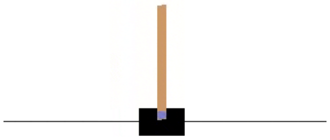
        - Reward = observed reward from BLEU + λ * KL divergence(soft distribution model output, same soft distribution of RL output)
    - Decay λ and T so model is only rewarded by own decisions and is increasingly confident about predictions

  The intuition behind this approach is the model will begin relying more on its own predictions as the lambda factor is decayed. There are two issues with this approach, the first is the RL model performance is very sensitive to the schedule of lambda and initial contribution factor and more second more important issue is the supervised model suffers from exposure bias. Therefore, rewarding the RL agent for mirroring potentially undesired predictions hurts learning and overall performance.

- **Pretraining the critic:** randomly sampling the supervised model as the actor and training the critic can be thought of as a form of knowledge transfer as it allows the critic to form a mapping between the states visited by a supervised model and their reward. Actions which vary very dramatically from the supervised model (very low probability actions) can be expected to achieve low rewards. As the Actor-Critic model learns through estimating the difference between the observed value and the expected value given this state, pretraining the critic based on the supervised model improves its performance.

CartPole Environment

Environment Description



CartPole (Barto et al., 1983) is one of the earliest reinforcement learning control problems and describes a system in which a pole is attached to a cart that can either move left or right and the objective is to balance the pole in the air for as long as possible. At each step in the episode, the agent receives the position of cart, velocity of cart, angle of pole, rotation rate of pole as continuous values and needs to decide whether to move left or right.

The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center (OpenAI Gym implementation). For each step the pole remains balanced, the agent receives one point (dense reward signal). Given the state space is continuous, solving

CartPole requires using either a function approximator or quantization. In spite of this, it is a fairly simple problem to solve and is often used as a first baseline task to ensure the code works properly.

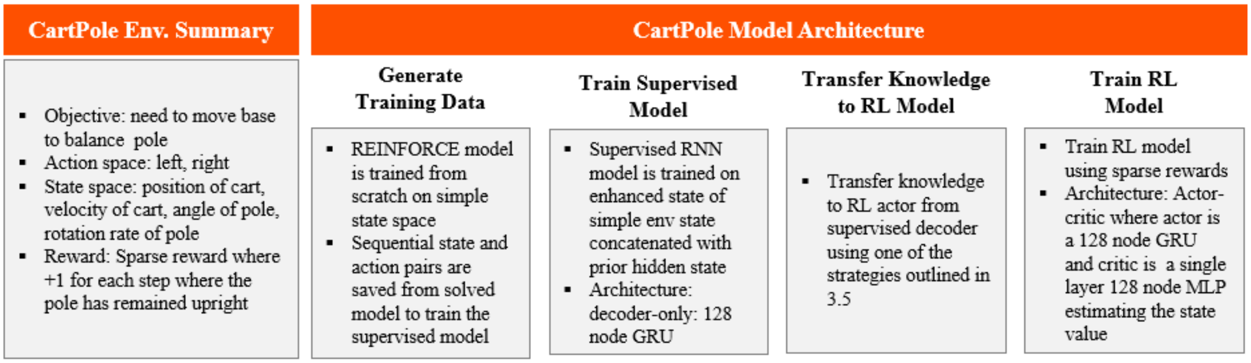
Modifications to Environment

As CartPole in its original form is a very different problem from the ParaPhrasee environment, several modifications were made. The most important is rather than using a dense reward signal, the agent only learns how many steps it has successfully taken at the end of the episode converting it to a sparse reward signal. This is intended to introduce the credit assignment problem and ends up being more challenging in terms of credit assignment than ParaPhrasee as the sequence extends over 200 steps vs ParaPhrasee's ~12.

While the problem can easily be solved using only the original 4 dimensions of the state space, to make the problem more similar to ParaPhrasee, we enrich the state space using the prior hidden state from the RNN model. This increases the size of the state space significantly and requires the model extract the relevant information in order to achieve a high return. The state space is also non-stationary as the hidden state representation will change as the model is trained. Capturing prior actions through the hidden state is unnecessary to solve this environment as the original state space is Markov and captures all relevant information from the history although makes the problem more similar to ParaPhrasee.

Model Architecture Description

In order to find an architecture which is expected to perform well on ParaPhrasee, we built architectures which can be expected to scale to more complex tasks. An Actor-Critic architecture for the reinforcement learning agent was followed in order to scale to ParaPhrasee. Given the simplicity of the problem, a REINFORCE network is first trained from scratch until it can solve the environment. The trained model is then used to create a supervised training set of sequential states and action pairs. A supervised RNN model is then trained (128 node GRU architecture with a log-softmax output layer, Adam optimizer, and negative log likelihood loss function) on this data. Then the knowledge from the supervised RNN model is transferred to the RL model with the same architecture using one of the knowledge transfer strategies discussed in 3.4. The following design choices were tested: the impact of transferring the weights vs training from scratch, using policy distillation, using a REINFORCE architecture, and using MLE rather than sampling.



CartPole Env. Summary

- Objective: need to move base to balance pole
- Action space: left, right
- State space: position of cart, velocity of cart, angle of pole, rotation rate of pole
- Reward: Sparse reward where +1 for each step where the pole has remained upright

CartPole Model Architecture

Generate Training Data

- REINFORCE model is trained from scratch on simple state space
- Sequential state and action pairs are saved from solved model to train the supervised model

Train Supervised Model

- Supervised RNN model is trained on enhanced state of simple env state concatenated with prior hidden state
- Architecture: decoder-only: 128 node GRU

Transfer Knowledge to RL Model

- Transfer knowledge to RL actor from supervised decoder using one of the strategies outlined in 3.5

Train RL Model

- Train RL model using sparse rewards
- Architecture: Actor-critic where actor is a 128 node GRU and critic is a single layer 128 node MLP estimating the state value

Implementation and Project Information

In order to achieve better repeatability within the RL research community, OpenAI's Gym package has implemented several common environments in Python (*https://gym.openai.com/docs/*). We leverage their implementation of CartPole although modify their code to remove rendering and change the reward function to be optionally sparse instead of always dense. The initial policy-gradient code is based on PyTorch's REINFORCE example code (*https://github.com/pytorch/examples/blob/master/reinforcement_learning/reinforce.py*). The Actor-Critic model is based on PyTorch's Actor-Critic example code (*https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py*).

FrozenLake Environment

Environment Description

FrozenLake is a classic path searching problem in which an agent is trying to navigate from a start space to the goal without falling into a hole. If the agent is able to get to the goal it achieves 20 points, if it falls in a hole it loses 10 and for each step it takes it loses 1 point. In the common formulation of FrozenLake, the environment is stochastic meaning if you select a certain action with a probability you will end up in a random different end state. Although at each episode the agent restarts at the start space. In most formulations, the map dynamics do not change (e.g. the probabilities remain stationary) therefore the agent can solve the environment through sufficient exploration as at each step it receives its location and can build a model of the environment.

S	F	F	H	F
F	F	H	F	F
F	F	F	F	F
F	F	H	F	F
H	F	F	F	G

Modifications to Environment

Reinforcement learning has long been applied to path searching problems with simple state spaces (Kaelbling et al., 1996). However, to make the problem more similar to ParaPhrasee, several modifications were made. The most important difference is the map dynamics change each time. The start and goal squares remain in the same location although at the end of the episode a new map is generated each time with at least one valid path. This means the agent can no longer solve the environment through exploration as the original state information (current location) is insufficient.

In order to make the problem similar to ParaPhrasee the agent is given a full map of the environment at time zero which it encodes into an embedding and needs to learn to use to take its next action. The enriched state the agent receives is its current position plus the hidden state from the prior RNN step which needs to learn to encode the map embedding it received at the start.

As ParaPhrasee is deterministic, the stochastic element of FrozenLake has been removed meaning if the agent takes an action it will always result in transitioning to the same next state. For the same reasons as the CartPole environment, FrozenLake has also been converted to sparse rewards where the agent only receives the return at the end of the episode. The dimensions of the map are 5x5 with a 75% chance a generated square is frozen such that at least one valid path is formed.

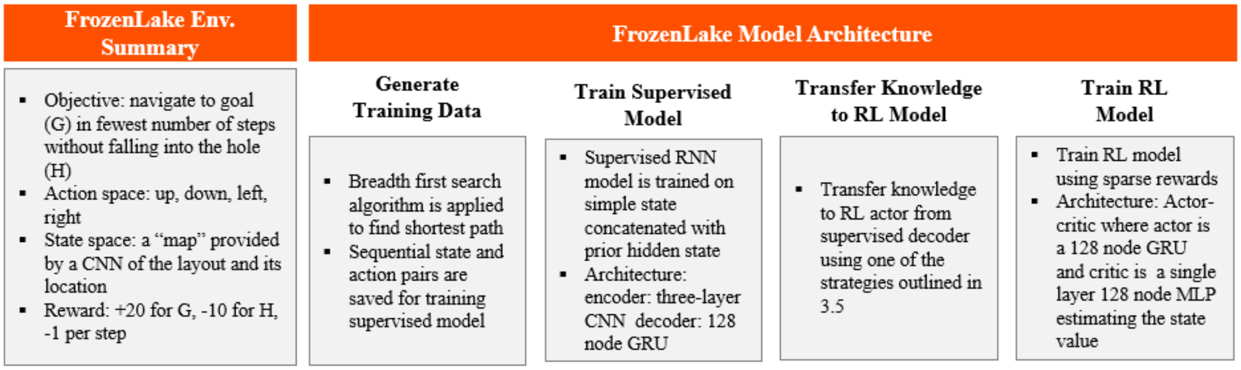
Model Architecture Description

As with the CartPole environment, in order to find an architecture which is expected to perform well on ParaPhrasee, we built architectures which can be expected to scale to more complex tasks. An Actor-Critic architecture was followed in order to scale to ParaPhrasee for the reinforcement learning model. In order to make the problem more comparable to ParaPhrasee an encoder is trained to encode the map information.

In order to generate the supervised data, a breadth-first search agent was developed which given a map finds the shortest path using the breadth-first search algorithm and then returns a training set consisting of an input map, a sequence of states, and a sequence of actions. A supervised encoder and decoder are then trained on this data. We tested two neural networks: an MLP and a CNN as encoders, and the CNN performed much better therefore it was used. The CNNs architecture consists of two convolutional layers each with a kernel size of 3, stride of 2, with one space of padding. Max pooling is performed after each convolutional filter and a linear output is used to return the encoding layer. The map encoding is then passed to an RNN decoder model (128 node GRU architecture) and trained end to end with a log-softmax output layer, Adam optimizer, and negative log likelihood loss function.

Then the knowledge is transferred to the RL model with the same architecture using one of the knowledge transfer strategies discussed in 3.4 in order to fine-tune its performance. The following design choices were tested: the impact of transferring the weights vs training from scratch, using policy distillation, using a REINFORCE architecture, and using MLE rather than sampling.

Two models were evaluated, a medium strength supervised encoder model and a strong supervised encoder model to determine how fine-tuning performs given different encoder models. We also test the performance of using Monte Carlo Tree Search (MCTS) on this environment once the models are fully trained. Implementing MCTS in a computationally efficient way in terms of memory and performance was an interesting challenge. Although modifying a vectorized implementation achieves a significant speedup versus more general MCTS packages for Python.



FrozenLake Env. Summary

- Objective: navigate to goal (G) in fewest number of steps without falling into the hole (H)
- Action space: up, down, left, right
- State space: a "map" provided by a CNN of the layout and its location
- Reward: +20 for G, -10 for H, -1 per step

FrozenLake Model Architecture

Generate Training Data

- Breadth first search algorithm is applied to find shortest path
- Sequential state and action pairs are saved for training supervised model

Train Supervised Model

- Supervised RNN model is trained on simple state concatenated with prior hidden state
- Architecture: encoder: three-layer CNN decoder: 128 node GRU

Transfer Knowledge to RL Model

- Transfer knowledge to RL actor from supervised decoder using one of the strategies outlined in 3.5

Train RL Model

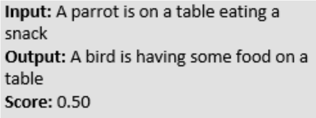
- Train RL model using sparse rewards
- Architecture: Actor-critic where actor is a 128 node GRU and critic is a single layer 128 node MLP estimating the state value

Implementation and Project Information

Similar to CartPole, FrozenLake is a common environment and is implemented in OpenAI gym. We began with their code and made several modifications including simplifying certain logic, adding a reset method, removing rendering logic, modifying the reward function, and handling object memory. Breadth-First Search is a very widely used algorithm for finding the shortest path and our implementation is based on this video *(https://www.youtube.com/watch?v=KiCBXu4P-2Y)*. The Actor-Critic model is based on PyTorch's Actor-Critic example code *(https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py)* and was adapted for FrozenLake. MCTS was relatively challenging to implement efficiently although we modified existing code to accommodate FrozenLake, to be able to do sample rollouts, and to better handle object memory *(https://github.com/brilee/python_uct/blob/master/numpy_impl.py)*.

ParaPhrasee Environment

Environment Description



Input: A parrot is on a table eating a snack
Output: A bird is having some food on a table
Score: 0.50

We developed the ParaPhrasee environment to represent the real paraphrase generation task. The ParaPhrasee environment handles encoding the input sentence into a sentence embedding, state transitions, calculating the reward relative to a reference sentence, and returning the generated sentence. Given it is a newly developed environment, multiple ways of formulating the paraphrase generation task as a RL problem were tested. The approach which we implemented was to treat the environment as similar to RNN formulation in which the state is the prior word (which is converted to a word embedding) and the decoder's hidden state at the last step. The episode starts with the <SOS> token and the encoded input sentence from the encoder and ends when the decoder generates a <EOS> token.

The reward functions tested were as follows: BLEU (n=1 & n=2), ROUGE, CIDEr, PARA, PARASIM, ESIM, and adversarial ESIM. BLEU, ROUGE, and CIDEr are explained in section 2.2 although PARA, PARASIM, ESIM and adversarial ESIM are new names for different evaluation approaches:

- **PARA:** is short for paraphrase similarity and is a metric in which checks the similarity between the input sentence and generated sentence embeddings. Both BERT and InferSent were tested in preliminary stages although BERT performed better therefore it was selected as the encoder for the main tests (explained in further detail in 2.2).

- **ESIM:** is a neural model which models local interactions between sentences to determine whether they are similar (Chen et al., 2017) (explained in further detail in 2.2). In order to train ESIM on identifying which sentences are paraphrases and which are not, we pretrain it on three sources of data in equal proportions.

    - *Correct paraphrase pairs*

    - Input sentence: a tennis player is swinging a racket at a ball.
    - Reference sentence: a man who is swinging a tennis racket.
    - Target value: Paraphrase
  - *Incorrect paraphrase pairs:* sentences pairs which have been randomly shuffled
    - Input sentence: a wicker basket carrying a variety of fruit.
    - Reference sentence: kids sleeping in a bed with fluffy blankets.
    - Target value: Non-paraphrase
  - *Noisy paraphrases:* with probability p (40% in experiments) the word is either copied from the input sentence or replaced with a random word sampled from the vocabulary based on its frequency count in the training set. This is to make the model more robust to sentences which contain significant word overlap although are not fluent. One downside to this approach is that it occasionally generates sentences which are considered paraphrases due to the words which are randomly substituted minorly impacting the semantic meaning.
    - Input sentence: high rise building with a blue sky in the background.
    - Reference sentence: high around building with bat blue sky selling the background.
    - Target value: Non-paraphrase

- **PARASIM:** A combination of PARA and ESIM metrics.

- **Fluency (F):** given a sentence want to evaluate how fluent it is. After researching existing fluency metrics, we were unable to find a suitable metric. As such, we use the normalized perplexity from a strong neural network trained on a large corpus of data as a proxy for fluency. We consider both GPT (Radford et al., 2018) and GPT-2 (Radford et al., 2018) as language models and get the normalized the perplexity score for a given sentence through the following equation: 1-min((perplexity/max_perplexity), 0.99) where max_perplexity is 500. This means that lower perplexity scores result in higher fluency and higher perplexity scores are capped at 500 which translates to a lower fluency score of 0.01. The downside to this approach is that perplexity is a very coarse proxy for fluency as a sentence such as "my name is Andrew" is likely to have a much lower perplexity score than "my name is Xenu" despite the fluency being identical. That being said, this approach is still good at penalizing sentences which are non-grammatical and addresses the issue in generation of word repetition such as “The dog is eating a a a a bone”.

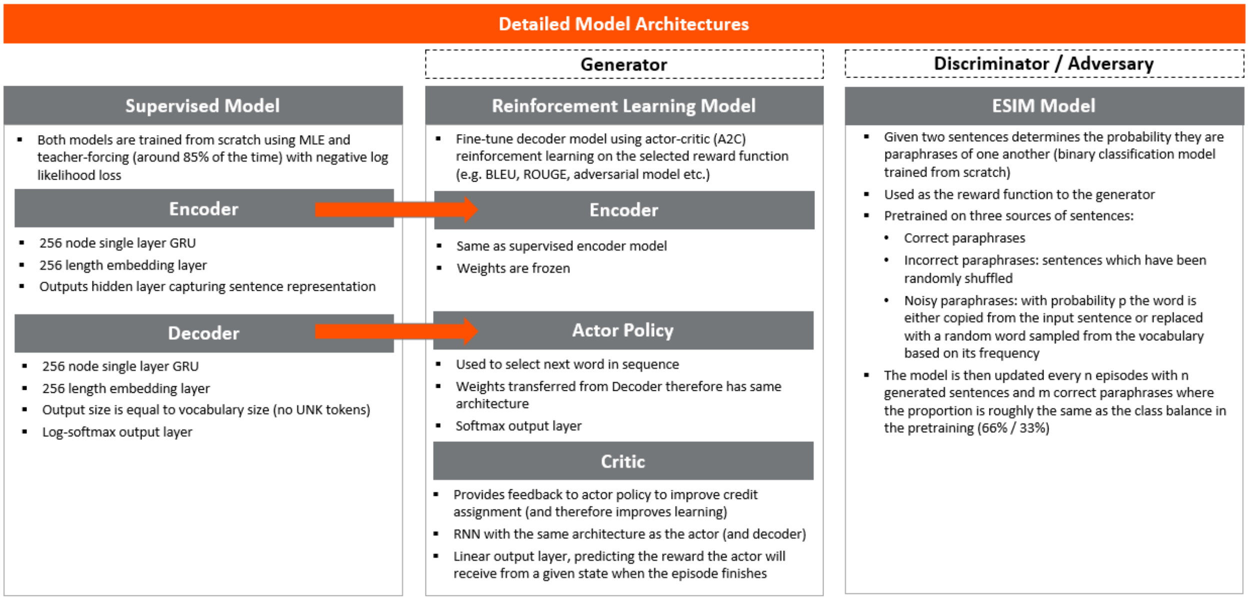
Combining metrics as seen with PARASIM was also heavily tested. Particularly, we sought to combine metrics for semantic relatedness such as PARA or ESIM with metrics for fluency. Multiple approaches were attempted including linear combinations with different weights, F-measure combinations with different betas, and scaling the data before weighting including applying standardization and calculating the cumulative distribution function for the evaluation metrics based on the sentence scores generated by the supervised model. Unfortunately, as discussed in the results section 4.2, these approaches were underwhelming and did not result in generalizable principles for scalarization.

Model Architecture Description

Given the ParaPhrasee environment’s significantly greater computational complexity than either CartPole or FrozenLake, rather than directly test all potential architecture configurations, we rely on applying approaches which performed well across both environments. The approach which worked the best is outlined below and is to select the best supervised encoder, use an Actor-Critic model with the same architecture as the supervised model and transfer the network weights to the RL decoder (the actor agent), randomly sample instead of using MLE for the actor during training, not use policy distillation, pretrain the critic using the supervised as an actor, and use stochastic gradient descent as an optimizer to fine-tune rather than Adam.

The neural architecture is as follows: the supervised encoder is a 256 node GRU with a 256-length word embedding vector for each token that returns its hidden state (also length 256) to the decoder which is an identical architecture except it outputs a log-softmax over the vocabulary. The RL actor has the same architecture as the decoder except the output is a softmax instead of log-softmax. The critic is another GRU with 256 nodes and takes environment state (the prior token and the actor’s hidden state) of and outputs the predicted return value for that state. Finally, the environment returns the reward when the generated sentence is completed, and the actor and critic models are updated although the encoder remains static.

When using an adversarial reward function, the process remains similar, although the key difference is the reward is determined by a network which itself updates every n episodes. We begin with the ESIM model which has been pretrained on the three sources of data outlined above then every 6000 episodes of updating generator (actor-critic model), the discriminator (ESIM model) is updated with 70% of samples randomly selected from the generated sentences and 30% randomly selected from the training set. The intention being to keep roughly the same class imbalance as the adversary was trained on. This repeats for m number of episodes (~30k). Both models are trained using online learning. Monte Carlo Tree Search is then used as a beam-search decoder once the models are trained to further improve performance.



Detailed Model Architectures

Supervised Model

- Both models are trained from scratch using MLE and teacher-forcing (around 85% of the time) with negative log likelihood loss

Encoder

- 256 node single layer GRU
- 256 length embedding layer
- Outputs hidden layer capturing sentence representation

Decoder

- 256 node single layer GRU
- 256 length embedding layer
- Output size is equal to vocabulary size (no UNK tokens)
- Log-softmax output layer

Generator

Reinforcement Learning Model

- Fine-tune decoder model using actor-critic (A2C) reinforcement learning on the selected reward function (e.g. BLEU, ROUGE, adversarial model etc.)

Encoder

- Same as supervised encoder model
- Weights are frozen

Actor Policy

- Used to select next word in sequence
- Weights transferred from Decoder therefore has same architecture
- Softmax output layer

Critic

- Provides feedback to actor policy to improve credit assignment (and therefore improves learning)
- RNN with the same architecture as the actor (and decoder)
- Linear output layer, predicting the reward the actor will receive from a given state when the episode finishes

Discriminator / Adversary

ESIM Model

- Given two sentences determines the probability they are paraphrases of one another (binary classification model trained from scratch)
- Used as the reward function to the generator
- Pretrained on three sources of sentences:
  - Correct paraphrases
  - Incorrect paraphrases: sentences which have been randomly shuffled
  - Noisy paraphrases: with probability p the word is either copied from the input sentence or replaced with a random word sampled from the vocabulary based on its frequency
- The model is then updated every n episodes with n generated sentences and m correct paraphrases where the proportion is roughly the same as the class balance in the pretraining (66% / 33%)

MCTS is applied to final decoder model to further improve performance

Implementation and Project Information

The encoder is from the supervised model, so we leveraged the same base code. InferSent was used out-of-the-box from the open-source code after making minor tweaks in order to run it (*https://github.com/facebookresearch/InferSent*). BERT requires fine-tuning to produce higher quality sentence embeddings. Therefore we used the sentence-transformers package which fine-tunes BERT using a Siamese network structure to produce semantically useful sentence embeddings (*https://pypi.org/project/sentence-transformers/*). For fluency we rely on HuggingFace's PyTorch-Transformers package for GPT, and GPT-2 implementations (*https://github.com/huggingface/transformers*). We modify of Lan & Xu's (2018) ESIM model implementation through updating the code for our current PyTorch version and modifying it to work with our dataset. The Actor-Critic model is based on PyTorch's Actor-Critic example code *(https://github.com/pytorch/examples/blob/master/reinforcement_learning/actor_critic.py)* and was adapted to the ParaPhrasee environment and to increase flexibility. Monte Carlo Tree Search follows the same implementation as FrozenLake.

Coco API's evaluation metrics were used *(https://github.com/tylin/coco-caption)* to ensure consistency in the evaluating the reported metrics with other prior work. As the package itself is a combination of many other authors' work, there was a significant amount of work in making it run in Python 3 across Windows and Linux as it also relies on Java. The code was designed for batch evaluation against a formatted JSON file; therefore, it was necessary to convert it to single sentence pair evaluation.

4 Results

This section discusses the product of applying the methods in order to answer the research questions. It is divided into three sections; the supervised learning paraphrase generation results, the intermediary reinforcement learning environment results, and the results achieved on the main reinforcement learning paraphrase generation task.

4.1 Supervised Learning Paraphrase Generation Results

One of the secondary research questions asks, “What is the best sequence to sequence architecture for paraphrase generation?”. In order to test this, we evaluated several common architectures as discussed in section 3.4 with different hyperparameters.

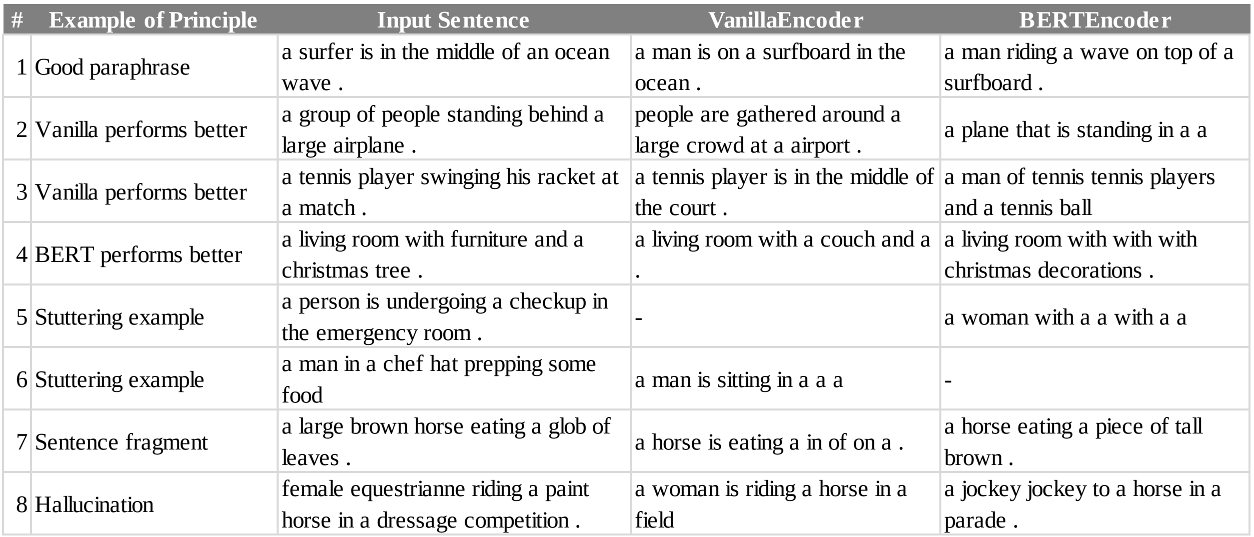
While improved performance on negative log likelihood loss correlates with improved performance on the evaluation metrics and text generation quality, models with higher validation error may still achieve better generation quality due to overfitting on the validation set. As a result of this, we test three models for key model configurations with low validation error and select the best one to evaluate that configurations performance.

Supervised Learning Test Set Evaluation Performance

#	Experiment Name	BLEU-1	BLEU-2	ROUGE	PARA	Fluency	ESIM	Average	N Iterations	N Epochs	Start TF Ratio	End TF Ratio	Optimizer
Vanilla Encoder / Decoder													
1	SGD Optimizer												
	Validation Loss: 3.118	0.3439	0.1471	0.3404	0.6469	0.6238	0.6079	0.4517	30	5000	0.90	0.85	SGD
	Validation Loss: 3.139	0.3535	0.1509	0.3478	0.6535	0.7165	0.6866	0.4848	30	5000	0.90	0.85	SGD
	Validation Loss: 3.150	**0.3612**	**0.1576**	**0.3515**	**0.6602**	**0.7380**	**0.6902**	**0.4931**	30	5000	0.90	0.85	SGD
2	Adam Optimizer (3.381 Loss)	0.3688	0.1622	0.3657	0.6543	0.6589	0.6271	0.4728	30	5000	0.90	0.85	Adam
3	Optimizer Switch (3.433 Loss)	0.3707	0.1556	0.3666	0.6450	0.7253	0.6544	0.4863	30	5000	0.90	0.85	Switch
4	Teacher Forcing (3.512 Loss)	0.3281	0.1308	0.3259	0.6101	0.3451	0.4310	0.3618	30	5000	0.90	0.50	SGD
Attention Encoder / Decoder													
5	SGD Optimizer (3.094 Loss)	0.3426	0.1541	0.3406	0.6654	0.7336	0.7165	0.4921	30	5000	0.90	0.85	SGD
6	Adam Optimizer (3.458 Loss)	0.3655	0.1592	0.3628	0.6562	0.6757	0.6379	0.4762	30	5000	0.90	0.85	Adam
Pretrained Encoder / Vanilla Decoder													
7	**Mean-Pooled GloVe Embeddings**												
	Validation Loss: 3.062	0.3535	0.1507	0.3515	0.6507	0.6306	0.5996	0.4561	30	5000	0.90	0.85	SGD
	Validation Loss: 3.124	**0.3635**	**0.1508**	**0.3569**	**0.6665**	**0.7390**	**0.6482**	**0.4875**	30	5000	0.90	0.85	SGD
	Validation Loss:3.131	0.3512	0.1430	0.3435	0.6512	0.7389	0.6429	0.4784	30	5000	0.90	0.85	SGD
8	**InferSent Encoder**												
	Validation Loss: 3.214	**0.3505**	**0.1404**	**0.3435**	0.6673	**0.7510**	0.7064	**0.4932**	30	5000	0.90	0.85	SGD
	Validation Loss: 3.244	0.3376	0.1396	0.3289	0.6587	0.7246	**0.7228**	0.4853	30	5000	0.90	0.85	SGD
	Validation Loss: 3.274	0.3194	0.1372	0.3125	**0.6700**	0.6772	0.7102	0.4711	30	5000	0.90	0.85	SGD
9	**BERT Encoder**												
	Validation Loss: 3.114	0.3490	0.1443	0.3448	**0.6796**	**0.7698**	**0.7144**	**0.5003**	30	5000	0.90	0.85	SGD
	Validation Loss: 3.149	0.3414	0.1368	0.3335	0.6585	0.7005	0.6660	0.4728	30	5000	0.90	0.85	SGD
	Validation Loss: 3.161	**0.3530**	**0.1482**	**0.3461**	0.6783	0.7135	0.6860	0.4875	30	5000	0.90	0.85	SGD

The best performing model based on average performance across all metrics is the BERT encoder. The difference in performance is however fairly negligible between Vanilla Encoder with SGD, BERT, and InferSent despite BERT and InferSent requiring a much greater amount of computation to encode and larger model sizes due to the larger embeddings. In addition to evaluating the models based on their test set performance on the above evaluation metrics, we also consider the qualitative generation performance on the generated results (sample contained in Appendix C).

Examples of Generated Sentences (using MLE models)



#	Example of Principle	Input Sentence	VanillaEncoder	BERTEncoder
1	Good paraphrase	a surfer is in the middle of an ocean wave .	a man is on a surfboard in the ocean .	a man riding a wave on top of a surfboard .
2	Vanilla performs better	a group of people standing behind a large airplane .	people are gathered around a large crowd at a airport .	a plane that is standing in a a
3	Vanilla performs better	a tennis player swinging his racket at a match .	a tennis player is in the middle of the court .	a man of tennis tennis players and a tennis ball
4	BERT performs better	a living room with furniture and a christmas tree .	a living room with a couch and a .	a living room with with with christmas decorations .
5	Stuttering example	a person is undergoing a checkup in the emergency room .	-	a woman with a a with a a
6	Stuttering example	a man in a chef hat prepping some food	a man is sitting in a a a	-
7	Sentence fragment	a large brown horse eating a glob of leaves .	a horse is eating a in of on a .	a horse eating a piece of tall brown .
8	Hallucination	female equestrianne riding a paint horse in a dressage competition .	a woman is riding a horse in a field	a jockey jockey to a horse in a parade .

The Vanilla Encoder with SGD generalizes better than BERT qualitatively although both models have many examples of **stuttering** (repeating the same word several times), **generating sentence fragments** (incomplete and ungrammatical sentences) and **hallucinations** (adding detail not in the input sentence). These are the most common errors and are commonly faced throughout NLG (Xie, 2018). Using beam search and fine-tuning using reinforcement learning can improve the performance of both of these approaches.

Variations to the VanillaEncoder including using the Adam optimizer, switching optimizers, and different rates of teacher forcing all resulted in lower performance than simply using SGD. However, Adam and Optimizer Switch are both comparable in performance and result in much lower training times which could be advantageous depending on the application. Ending the teacher forcing rate at 0.50 instead of 0.85 resulted in a significant deterioration in performance with the worst results of any configuration. The attention model using SGD resulted in similar performance to VanillaEncoder and is likely to outperform if tuned properly as will be explored in future work.

Evaluation Metrics Correlation Matrix (VanillaEncoder SGD 3.150)

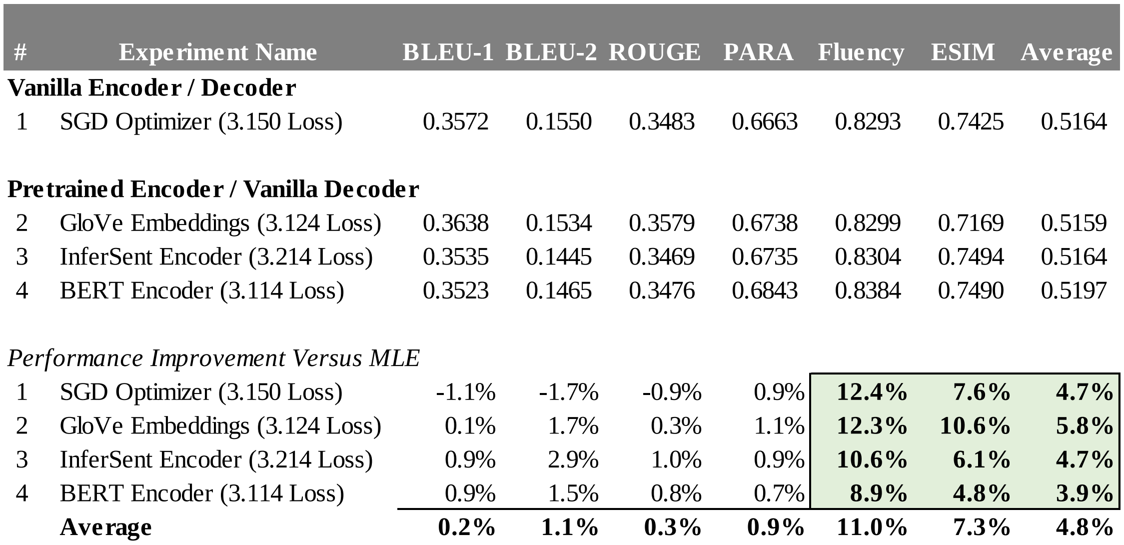
	BLEU1	BLEU2	ROUGE	PARA	Fluency	ESIM
BLEU1	-	0.79	0.94	0.45	0.10	0.16
BLEU2	0.79	-	0.81	0.43	0.12	0.17
ROUGE	0.94	0.81	-	0.44	0.09	0.16
PARA	0.45	0.43	0.44	-	0.19	0.57
Fluency	0.10	0.12	0.09	0.19	-	0.43
ESIM	0.16	0.17	0.16	0.57	0.43	-

The correlation between evaluation metrics is also important to consider in model selection. The correlation between metrics was calculated using the evaluation metrics on the test for the VanillaEncoder w/ SGD using MLE. Unsurprisingly, BLEU1 and ROUGE show very high correlation as the sentences are both roughly equal length and very short. PARA showed a surprisingly high correlation between all other metrics except Fluency. The lack of correlation between Fluency and BLEU1, BLEU2 and ROUGE highlight why word level metrics are fundamentally challenged as automatic evaluation metrics. ESIM and PARA are quite correlated which is unintuitive as they are fairly different models. This may be due to both being attention-based approaches for similarity.

Impact of Using Beam Search

Using beam search for decoding the sentences considerably improves the performance of the supervised models. We use a beam size of 4 to balance computation and performance improvement given the max sentence size is 12.

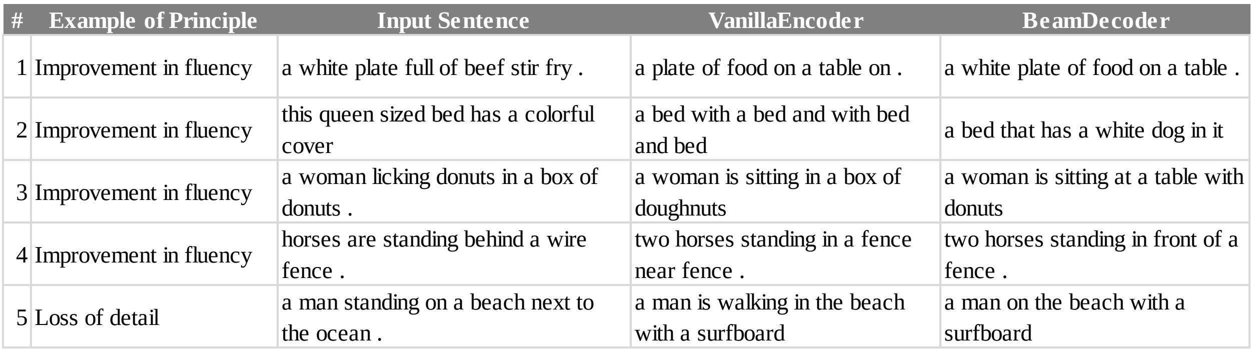
Beam Search Decoding Performance on Test Set



#	Experiment Name	BLEU-1	BLEU-2	ROUGE	PARA	Fluency	ESIM	Average
Vanilla Encoder / Decoder								
1	SGD Optimizer (3.150 Loss)	0.3572	0.1550	0.3483	0.6663	0.8293	0.7425	0.5164
Pretrained Encoder / Vanilla Decoder								
2	GloVe Embeddings (3.124 Loss)	0.3638	0.1534	0.3579	0.6738	0.8299	0.7169	0.5159
3	InferSent Encoder (3.214 Loss)	0.3535	0.1445	0.3469	0.6735	0.8304	0.7494	0.5164
4	BERT Encoder (3.114 Loss)	0.3523	0.1465	0.3476	0.6843	0.8384	0.7490	0.5197
Performance Improvement Versus MLE								
1	SGD Optimizer (3.150 Loss)	-1.1%	-1.7%	-0.9%	0.9%	**12.4%**	**7.6%**	**4.7%**
2	GloVe Embeddings (3.124 Loss)	0.1%	1.7%	0.3%	1.1%	**12.3%**	**10.6%**	**5.8%**
3	InferSent Encoder (3.214 Loss)	0.9%	2.9%	1.0%	0.9%	**10.6%**	**6.1%**	**4.7%**
4	BERT Encoder (3.114 Loss)	0.9%	1.5%	0.8%	0.7%	**8.9%**	**4.8%**	**3.9%**
	Average	**0.2%**	**1.1%**	**0.3%**	**0.9%**	**11.0%**	**7.3%**	**4.8%**

While all metrics show some improvement on average, Fluency and ESIM show the most significant improvements in performance. The improvement in Fluency does not necessarily result in an improvement in semantic relatedness as details are often added or the sentence is simplified.

Examples of Generated Sentences (Using Beam Decoding)



#	Example of Principle	Input Sentence	VanillaEncoder	BeamDecoder
1	Improvement in fluency	a white plate full of beef stir fry .	a plate of food on a table on .	a white plate of food on a table .
2	Improvement in fluency	this queen sized bed has a colorful cover	a bed with a bed and with bed and bed	a bed that has a white dog in it
3	Improvement in fluency	a woman licking donuts in a box of donuts .	a woman is sitting in a box of doughnuts	a woman is sitting at a table with donuts
4	Improvement in fluency	horses are standing behind a wire fence .	two horses standing in a fence near fence .	two horses standing in front of a fence .
5	Loss of detail	a man standing on a beach next to the ocean .	a man is walking in the beach with a surfboard	a man on the beach with a surfboard

While beam decoding does result in an improvement in performance, its performance is insufficient for fully automated commercial products although it could be used to generate suggestions for a human who can then edit. Another significant problem is the generation is uncontrolled meaning the generated text is not optimized for any particular objective.

Summary of Key Findings from Supervised Learning Models

The key findings from the supervised learning model experiments are:

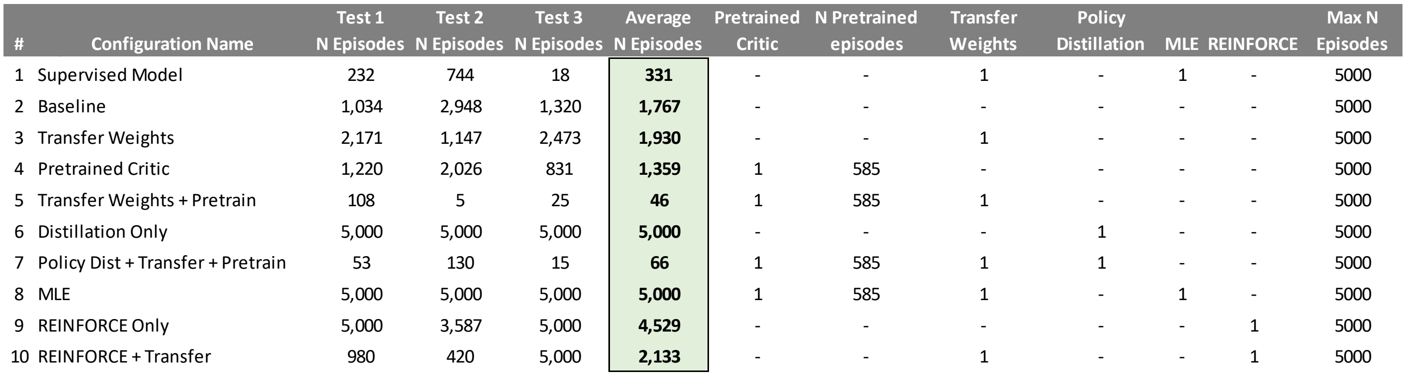
- Training a supervised model using MLE does a reasonable job generating paraphrases.
- There are common mistakes including stuttering, generating sentence fragments, and hallucinating details across all encoder models.
- While using BERT as an encoder achieves the best average performance, the difference is not substantial enough to warrant the incremental computation versus the VanillaEncoder.
- Training using teacher forcing is important to performance despite the exposure bias.
- BLEU1 and ROUGE are highly correlated given the short length generated sentences and target sentences.
- Using beam decoding instead of MLE on the trained model improves fluency dramatically at the expense of producing sentences which are more generic.

4.2 Intermediary Reinforcement Learning Environment Results

CartPole Environment

As discussed in section 3.5, CartPole is a fairly simple environment although it has been modified to make it more challenging and comparable with the ParaPhrasee environment. The main change is instead of providing a dense reward (return after each step), the environment uses a sparse reward signal with the agent receiving the return feedback at the end of the episode. This significantly changes the efficacy of the approaches and makes the results more generalizable to the ParaPhrasee environment.

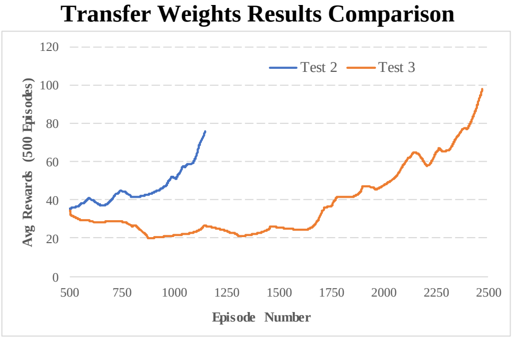
CartPole Environment Results



#	Configuration Name	Test 1 N Episodes	Test 2 N Episodes	Test 3 N Episodes	Average N Episodes	Pretrained Critic	N Pretrained episodes	Transfer Weights	Policy Distillation	MLE	REINFORCE	Max N Episodes
1	Supervised Model	232	744	18	**331**	-	-	1	-	1	-	5000
2	Baseline	1,034	2,948	1,320	**1,767**	-	-	-	-	-	-	5000
3	Transfer Weights	2,171	1,147	2,473	**1,930**	-	-	1	-	-	-	5000
4	Pretrained Critic	1,220	2,026	831	**1,359**	1	585	-	-	-	-	5000
5	Transfer Weights + Pretrain	108	5	25	**46**	1	585	1	-	-	-	5000
6	Distillation Only	5,000	5,000	5,000	**5,000**	-	-	-	1	-	-	5000
7	Policy Dist + Transfer + Pretrain	53	130	15	**66**	1	585	1	1	-	-	5000
8	MLE	5,000	5,000	5,000	**5,000**	1	585	1	-	1	-	5000
9	REINFORCE Only	5,000	3,587	5,000	**4,529**	-	-	-	-	-	1	5000
10	REINFORCE + Transfer	980	420	5,000	**2,133**	-	-	1	-	-	1	5000

The results for CartPole are summarized in the figure above. As CartPole is still a simple environment even with using a sparse reward signal, the relevant metric is how many episodes it takes to solve the environment rather than using the average score at convergence. Solving the environment is defined as balancing the pole for more than 295 steps on average across the last three episodes. As there is considerable variability in performance, the average of three tests was calculated using different random seeds to determine which configuration results in the best performance.

- **Supervised Model:** measures the performance achieved through simply using the trained weights and MLE to get a baseline performance of the supervised model before any fine-tuning using RL.
- **Baseline Model:** measures the performance training an Actor-Critic reinforcement model from randomly initialized weights to get a baseline for RL model performance.
- **Transfer Weights:** measures the impact of transferring the weights from the pretrained supervised model to the RL model to warm-start training. Interestingly, the performance decreases as a result of transferring the weights. This appears to result from the critic providing poor feedback to the actor which causes it to modify its policy away from its superior transferred policy. If the learning trajectory of the best performing Transfer Weights configuration (test 2) is compared with the worst trajectory (test 3), we can see that the performance decreases initially for test 3 although improves far more monotonically for test 2.



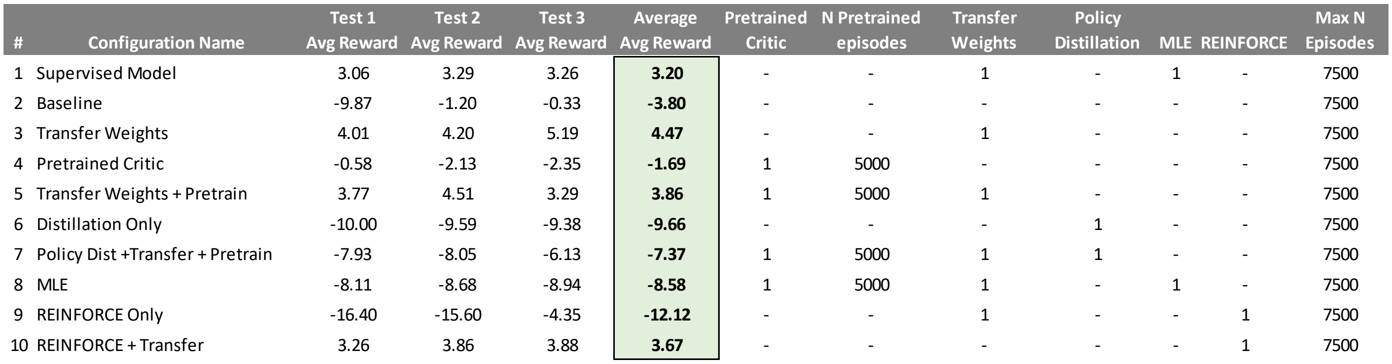
- **Pretrained Critic:** measures the impact of pretraining the critic using the supervised model as an actor for 585 episodes. The intuition of this test is to remedy the problem faced in config. 2 where the critic itself is learning how to estimate the value function and returns noisy feedback to the actor which impedes learning. As expected, this improves performance which is consistent with prior work (Zhang & Ma, 2018).

- **Transfer Weights + Pretrain:** through doing both transferring weights from the actor and pretraining the critic, we achieve the best results of any configuration. Most importantly, this shows a dramatic improvement over the supervised model demonstrating the efficacy of fine-tuning using RL.
- **Distillation Only:** the main purpose of distillation approaches is to efficiently transfer knowledge from one model to another which enables the second model to have a different architecture. Distillation alone performed very poorly being unable to solve the environment in the maximum number of episodes provided across all three tests. While it is likely that there exists a combination of weights and a decay schedule which would result in improved performance, we were unable to find an approach which worked consistently and therefore it is extremely unlikely it would generalize to other environments.
- **Policy Distillation + Transfer + Pretrain:** the addition of transferring the weights and pretraining resulted in much stronger performance although still worse than transferring the weights and pretraining without distillation. Given the weight placed on the KL divergence from the supervised model is inversely proportional to the RL model's performance, if the model already has strong performance then the penalty is relatively low so it would have a smaller impact on learning.
- **MLE:** rather than sample the distribution to select actions, MLE only takes the maximum likelihood probability at each step while training. Even with pretraining and transferring the weights this results in poor performance not being able to converge in the maximum number of steps.
- **REINFORCE Only & REINFORCE + Transfer:** using the REINFORCE algorithm as an RL agent instead of Actor-Critic results in decreased performance when training from scratch and also when transferring the weights from the supervised model.

FrozenLake Environment

FrozenLake is more comparable to ParaPhrasee as discussed in section 3.5, given it requires the agent maintain a representation of the encoder state to guide its action selection. FrozenLake is used to compare the impact on performance of using a medium strength pretrained supervised encoder vs using a strong pretrained encoder. Similar to CartPole, it is also a sparse environment although it has a slightly larger action space (4 actions).

FrozenLake Environment Results | Medium Encoder



#	Configuration Name	Test 1 Avg Reward	Test 2 Avg Reward	Test 3 Avg Reward	Average Avg Reward	Pretrained Critic	N Pretrained episodes	Transfer Weights	Policy Distillation	MLE	REINFORCE	Max N Episodes
1	Supervised Model	3.06	3.29	3.26	**3.20**	-	-	1	-	1	-	7500
2	Baseline	-9.87	-1.20	-0.33	**-3.80**	-	-	-	-	-	-	7500
3	Transfer Weights	4.01	4.20	5.19	**4.47**	-	-	1	-	-	-	7500
4	Pretrained Critic	-0.58	-2.13	-2.35	**-1.69**	1	5000	-	-	-	-	7500
5	Transfer Weights + Pretrain	3.77	4.51	3.29	**3.86**	1	5000	1	-	-	-	7500
6	Distillation Only	-10.00	-9.59	-9.38	**-9.66**	-	-	-	1	-	-	7500
7	Policy Dist +Transfer + Pretrain	-7.93	-8.05	-6.13	**-7.37**	1	5000	1	1	-	-	7500
8	MLE	-8.11	-8.68	-8.94	**-8.58**	1	5000	1	-	1	-	7500
9	REINFORCE Only	-16.40	-15.60	-4.35	**-12.12**	-	-	1	-	-	1	7500
10	REINFORCE + Transfer	3.26	3.86	3.88	**3.67**	-	-	-	-	-	1	7500

The results for FrozenLake using a pretrained supervised encoder which has been trained to medium performance (fewer epochs) are summarized in the figure above. As FrozenLake is a more complex environment than CartPole, the relevant metric is how strong the average performance is across the last n episodes given the same number of training episodes. As there is considerable variability in performance, the average of three tests was calculated using different random seeds to determine which configuration results in the best performance.

- **Supervised Model:** measures the performance achieved through simply using the trained weights and MLE to get a baseline performance of the supervised model before any fine-tuning using RL.

- **Baseline Model:** measures the performance training an Actor-Critic reinforcement model from randomly initialized weights to get a baseline for RL model performance.

- **Transfer Weights:** measures the impact of transferring the weights from the pretrained supervised model to the RL model to warm-start training. This results in the best performance out of any of the approaches which is counterintuitive as it is expected that pretraining the critic would be additive to performance as discussed in the Pretrained Critic bullet below.

- **Pretrained Critic:** measures the impact of pretraining the critic using the supervised model as an actor for 5,000 episodes. While this improves performance over the baseline results, the agent's performance is still negative at 7,500 episodes. This is likely due to pretraining the critic for too few episodes.

- **Transfer Weights + Pretrain:** through both transferring weights from the actor and pretraining the critic, we achieve the second-best results of the tested configurations.
- **Distillation Only:** the main purpose of distillation approaches is to efficiently transfer knowledge from one model to another which enables the second model to have a different architecture. Distillation alone performed very poorly with the second worst results of the tested configurations.
- **Policy Distillation + Transfer + Pretrain:** the addition of transferring the weights and pretraining resulted in slightly stronger performance although still much worse than the baseline RL model.
- **MLE:** rather than sample the distribution to select actions, MLE only takes the maximum likelihood probability at each step while training. Even with pretraining and transferring the weights this results in poor performance.
- **REINFORCE Only & REINFORCE + Transfer:** using the REINFORCE algorithm as an RL agent instead of Actor-Critic results in the worst performance of any configuration when trained from scratch. However, when transferring the weights from the supervised model the performance is relatively strong and surpasses the supervised model and falls roughly in line with the Actor-Critic model with pretraining and weight transfer (configuration 5).

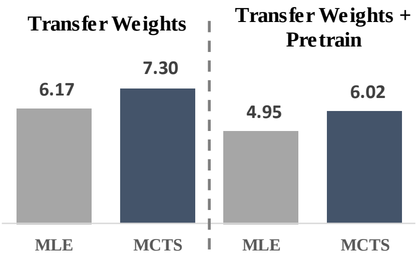
Results from Applying Monte Carlo Tree Search (MCTS)

MCTS is a model-based planning algorithm in which the agent samples actions in multiple roll-outs and determines which path results in the best reward. We test the impact of using MCTS instead of simply selecting the most likely action for the two top performing models using a medium strength encoder (Transfer Weights only and Transfer Weights + Pretrain Critic).

As MCTS takes a long time to complete its rollouts we only do a rollout when the agent's prediction confidence falls below a given threshold. For these experiments we selected a threshold of 0.90 meaning if the maximum probability selecting an action in a state is below 0.90 then the agent applies the MCTS algorithm and selects the action with the greatest expected return.

As seen in the figure to the right, using MCTS instead of maximum likelihood increases the performance for both Transfer Weights Only and Transfer Weights + Pretrain approaches however at the expense of longer inference time.

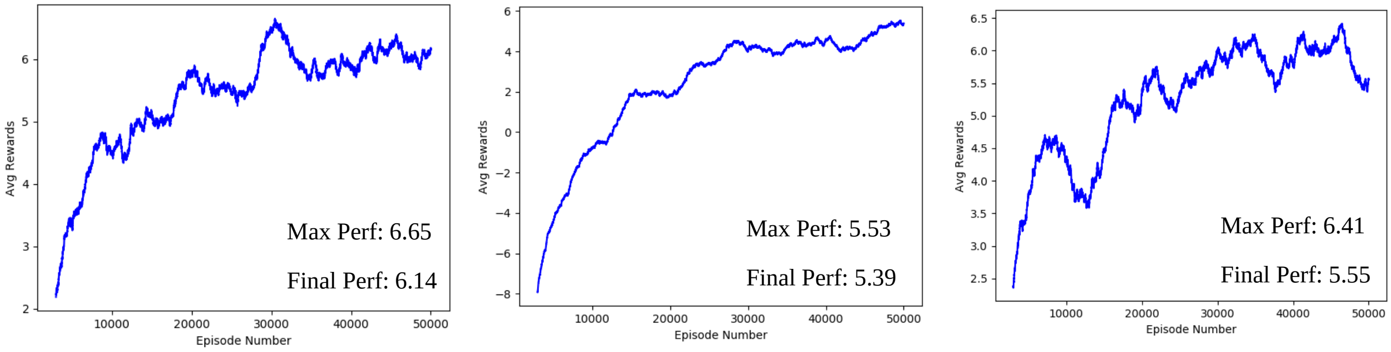
The performance improvement is fairly considerable with over a full point increase for each model without any incremental training. The performance would likely improve further with more simulations at each rollout. Whether to use MCTS or simply MLE depends on the use case given the greater compute requirements.


MCTS Results
(1,500 Episode Avg Reward)

Training Models for a Longer Duration

Due to computational constraints, each FrozenLake architecture configuration is only compared at 7,500 episodes which is sufficient to determine which models are likely to achieve greater performance if trained for longer. In order to determine how the models perform at convergence, we run three of the promising configurations (Transfer Weights, Pretrained Critic, and Transfer Weights + Pretrain Critic) to 50,000 episodes.

Model Performance at 50k Episodes | *3,000 Episode Moving Average*

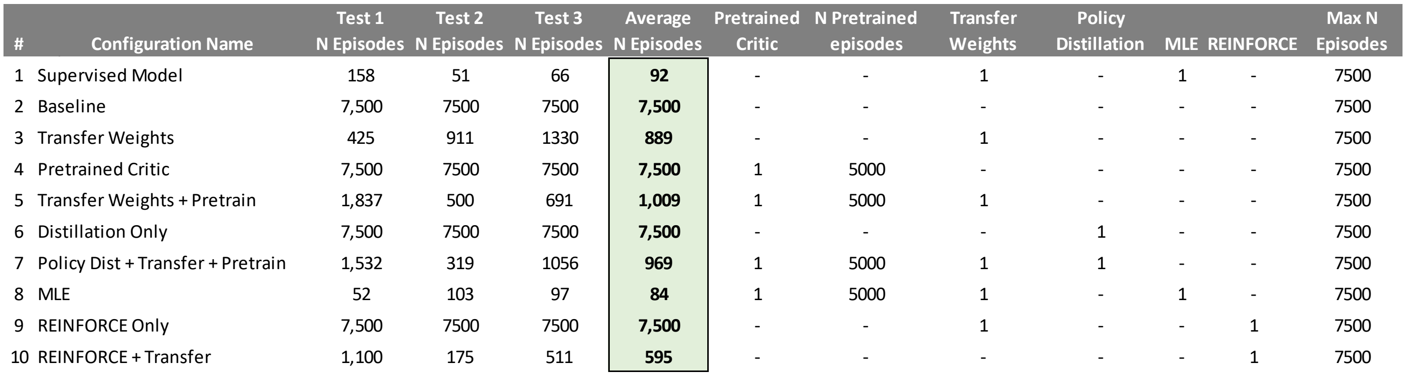

Transfer Weights **Pretrained Critic** **Transfer Weights + Pretrain**

The charts above show that each of the configurations are able to achieve performance which continues to improve with additional training episodes. Although the results are still noisy, all three approaches achieved roughly the same performance after 50k episodes which is significantly higher than the supervised model performance of ~3.20. The pretrained critic shows the least variation in rewards which can be attributed to the lack of bias from transferring the weights from the supervised model. It does still however achieve the lowest performance of the three.

Using a Strong Supervised Encoder as a Base Model

Rather than using a medium strength encoder, we investigate what the impact of using a strong supervised model as an encoder (one which is trained on a much greater number of epochs) is on the incremental performance an agent can achieve.

FrozenLake Environment Results | Strong Encoder



#	Configuration Name	Test 1 N Episodes	Test 2 N Episodes	Test 3 N Episodes	Average N Episodes	Pretrained Critic	N Pretrained episodes	Transfer Weights	Policy Distillation	MLE	REINFORCE	Max N Episodes
1	Supervised Model	158	51	66	**92**	-	-	1	-	1	-	7500
2	Baseline	7,500	7500	7500	**7,500**	-	-	-	-	-	-	7500
3	Transfer Weights	425	911	1330	**889**	-	-	1	-	-	-	7500
4	Pretrained Critic	7,500	7500	7500	**7,500**	1	5000	-	-	-	-	7500
5	Transfer Weights + Pretrain	1,837	500	691	**1,009**	1	5000	1	-	-	-	7500
6	Distillation Only	7,500	7500	7500	**7,500**	-	-	-	1	-	-	7500
7	Policy Dist + Transfer + Pretrain	1,532	319	1056	**969**	1	5000	1	1	-	-	7500
8	MLE	52	103	97	**84**	1	5000	1	-	1	-	7500
9	REINFORCE Only	7,500	7500	7500	**7,500**	-	-	1	-	-	1	7500
10	REINFORCE + Transfer	1,100	175	511	**595**	-	-	-	-	-	1	7500

The results for solving FrozenLake given a strong encoder are above. The problem becomes much easier when given a strong encoder and as such we use the total number of episodes it takes to solve the environment rather than using the average score at convergence as we did with CartPole. Solving the environment means achieving a return greater than 11.0 over the last 50 episodes. Again, to reduce the variance, the performance across three tests is averaged.

The main insight from this set of experiments is the only model which can achieve performance greater than the supervised model is fine-tuning the RL model with MLE. Transferring the weights significantly improved the performance relative to using the baseline Actor-Critic model. However, it still underperformed simply applying the supervised model.

This experiment demonstrates the concept that when supervised learning already achieves strong performance, there is unlikely to be much incremental gain from fine-tuning using reinforcement learning. This is partially due to the implicit upper bound in performance for most problems.

Summary of Key Findings from Intermediary Environments

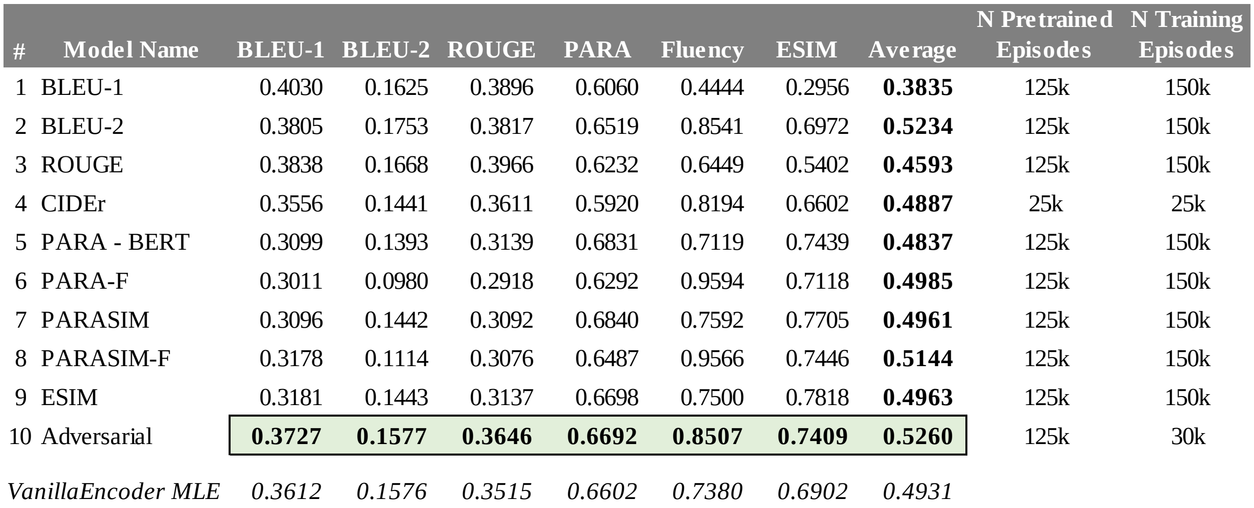
Based on the experiments on CartPole and FrozenLake, the following findings can be expected to generalize to ParaPhrasee and other similar environments:

- Actor-Critic outperforms REINFORCE as an RL algorithm and the extent to which it outperforms increases with the total number of steps per episode.
- Doing both transferring weights and pretraining the critic improves performance versus a randomly initialized Actor-Critic agent and compared to a supervised agent
    - Only transferring weights tends to improve the performance although it is dependent on how well the supervised model already performs and how challenging predicting the state-value is to learn for the Critic.
    - Only pretraining the critic improves the performance although the extent to which is proportional to the difficulty in predicting the state-value, the number of episodes of pretraining, and the extent to which the states visited by the RL model are similar to those visited by the supervised model.
    - The interaction between transferring the weights and pretraining the critic is non-linear. This is demonstrated by a ~38x improvement vs the baseline Actor-Critic when applying both for CartPole vs only transferring weights resulting in a deterioration in performance and only pretraining critic resulting in only a ~1.3x improvement.
    - Increasing training time until convergence tends to increase performance.
- The improvement in performance achieved from using reinforcement learning is dependant on the existing performance of the supervised model.
- Policy distillation is challenging to generalize to new environments and requires a considerable amount of hyperparameter tuning.
- There is considerable variability in the agent’s performance given by randomness in the agent’s weight initializations, in the environment resulting including differences in the sequence in which the agent visits certain states, and in the optimization process as we are using Adam.

4.3 Reinforcement Learning Paraphrase Generation Results

The main research question asks, “What is the most effective reinforcement learning reward function for paraphrase generation?” and in order to answer this we consider using common automatic evaluation metrics as reward functions, in addition to sentence level approaches, and finally adversarial approaches.

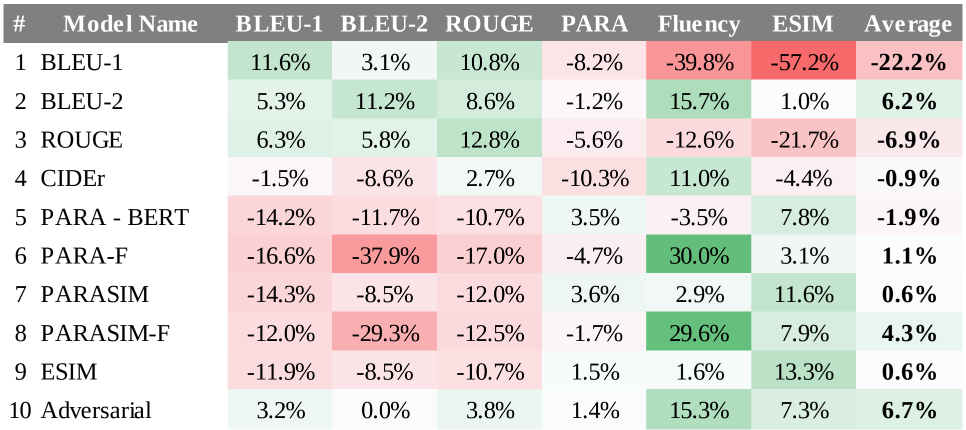
Reinforcement Learning Test Set Evaluation Performance



#	Model Name	BLEU-1	BLEU-2	ROUGE	PARA	Fluency	ESIM	Average	N Pretrained Episodes	N Training Episodes
1	BLEU-1	0.4030	0.1625	0.3896	0.6060	0.4444	0.2956	**0.3835**	125k	150k
2	BLEU-2	0.3805	0.1753	0.3817	0.6519	0.8541	0.6972	**0.5234**	125k	150k
3	ROUGE	0.3838	0.1668	0.3966	0.6232	0.6449	0.5402	**0.4593**	125k	150k
4	CIDEr	0.3556	0.1441	0.3611	0.5920	0.8194	0.6602	**0.4887**	25k	25k
5	PARA - BERT	0.3099	0.1393	0.3139	0.6831	0.7119	0.7439	**0.4837**	125k	150k
6	PARA-F	0.3011	0.0980	0.2918	0.6292	0.9594	0.7118	**0.4985**	125k	150k
7	PARASIM	0.3096	0.1442	0.3092	0.6840	0.7592	0.7705	**0.4961**	125k	150k
8	PARASIM-F	0.3178	0.1114	0.3076	0.6487	0.9566	0.7446	**0.5144**	125k	150k
9	ESIM	0.3181	0.1443	0.3137	0.6698	0.7500	0.7818	**0.4963**	125k	150k
10	Adversarial	**0.3727**	**0.1577**	**0.3646**	**0.6692**	**0.8507**	**0.7409**	**0.5260**	125k	30k
	VanillaEncoder MLE	*0.3612*	*0.1576*	*0.3515*	*0.6602*	*0.7380*	*0.6902*	*0.4931*		

The results in figures above and below show the performance of the various reinforcement learning approaches in absolute terms and relative to the supervised model which was used to warm-start all the RL agents.

Reinforcement Learning Relative Performance vs Vanilla Encoder w/ MLE



#	Model Name	BLEU-1	BLEU-2	ROUGE	PARA	Fluency	ESIM	Average
1	BLEU-1	11.6%	3.1%	10.8%	-8.2%	-39.8%	-57.2%	**-22.2%**
2	BLEU-2	5.3%	11.2%	8.6%	-1.2%	15.7%	1.0%	**6.2%**
3	ROUGE	6.3%	5.8%	12.8%	-5.6%	-12.6%	-21.7%	**-6.9%**
4	CIDEr	-1.5%	-8.6%	2.7%	-10.3%	11.0%	-4.4%	**-0.9%**
5	PARA - BERT	-14.2%	-11.7%	-10.7%	3.5%	-3.5%	7.8%	**-1.9%**
6	PARA-F	-16.6%	-37.9%	-17.0%	-4.7%	30.0%	3.1%	**1.1%**
7	PARASIM	-14.3%	-8.5%	-12.0%	3.6%	2.9%	11.6%	**0.6%**
8	PARASIM-F	-12.0%	-29.3%	-12.5%	-1.7%	29.6%	7.9%	**4.3%**
9	ESIM	-11.9%	-8.5%	-10.7%	1.5%	1.6%	13.3%	**0.6%**
10	Adversarial	3.2%	0.0%	3.8%	1.4%	15.3%	7.3%	**6.7%**

The impact on performance varies considerably across reward functions highlighting the importance of the main research question. While the metrics are unable to adequately capture the nuances of the generated text, they do illustrate the wider trends in the different approaches. Namely, optimizing for a specific metric will result in the highest performance on that metric and results in high performance on other metrics which correlate with that metric. While this aspect of the results in unsurprising, what is more interesting is the resulting impacts on the qualitative quality of the generated text and other metrics which are not correlated.

The RL agent often discovers an interesting strategy to maximize rewards which exploits the weaknesses in the evaluation metric rather than generating high quality text. This tends to result in poor performance on metrics which the agent is not directly optimizing.

Each metric is discussed individually below although the contrast between BLEU-1, PARASIM-F and the adversarial models best highlights the key themes throughout the section.

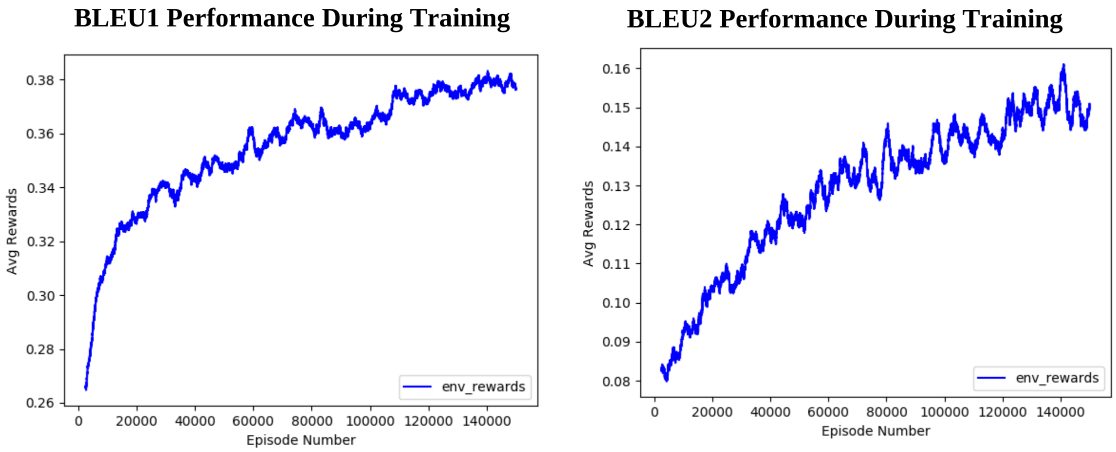
- When optimizing BLEU-1, we see a strong increase in word-level metrics at the expense of PARA, Fluency, and ESIM which captures the very poor generated sentence quality consisting of mostly filler words such as “the a on”.
- PARASIM-F represents a principled approach to balance semantic relatedness and fluency using an ensemble of three complex models to optimize all of PARA, Fluency and ESIM directly. However, the result is word-level metric performance suffers considerably and the agent finds a strategy to exploit weaknesses in the reward function.
- The adversarial approach, despite not having been given an explicit preference for any metric learns to optimize all metrics simultaneously and improves performance across each metric to achieve the best overall performance of any reward function.

BLEU

As discussed in section 2.2, BLEU is one of the most commonly used automatic evaluation metrics and therefore it is the first metric which we directly optimize. BLEU can be evaluated at multiple token levels. Given the short sentence size, we only consider BLEU-1 and BLEU-2. As BLEU does not have any measure of fluency and only operates at the word level, the model learns to include many connecting words such as “a is on the” for BLEU-1. Although the BLEU score is improved beyond the supervised model, the performance on metrics which consider fluency is very poor as the sentences are not coherent.

The performance using BLEU-2 has considerably better scores on the other metrics as increasing the n-gram size results in more likely word combinations. It is however noisier given the task is more challenging as the model needs to correctly predict tuples in the target sentence. As a result, the rewards fluctuate more significantly during training as seen in the figure below.

Comparison of Reward Metric Training Performance (2,500 Ep Avg)



BLEU1 Performance During Training

BLEU2 Performance During Training

BLEU was not designed for performance at the sentence level and therefore it was expected that the performance would be poor in this setting. The generated sentences highlight BLEU's inadequacy as an automatic evaluation metric.

Example generated sentences:

#	Input Sentence	MLE Sentence	BLEU1 Sentence	BLEU2 Sentence
1	a single sheep grazes in a field of grass .	a large sheep is standing in a field	a sheep is in a of a grass .	a sheep standing in a field with a .
2	a couple of large beds in a hotel room	a large room with beds in a room .	a room with a bed in a .	a room with a bed in a room .
3	a man in red shirt doing a trick on skateboard .	a skateboarder is doing a trick on a ramp .	a man is a a on in a .	a man on a skateboard on a skateboard .
4	wild red hair rocking out to guitar music .	a man is holding a in a the .	a man sitting on a bed in a .	a man sitting on a bed in a room .

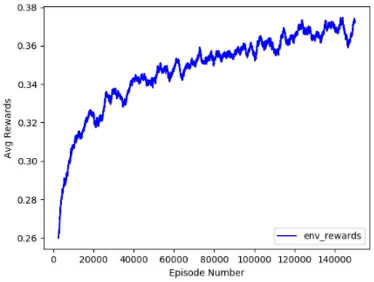
ROUGE

The impact on generation of using ROUGE as a reward metric is very similar to that of BLEU-1 as they are ~0.94 correlated given the short sentence sizes. Interestingly, optimizing ROUGE did not have the same detrimental impact on fluency or ESIM as it is recall based therefore there is less reward for including linking words. The fluency and ESIM scores are still quite a bit lower than BLEU-2 as ROUGE faces the same problems as BLEU-1.

Example Generated Sentences

#	Input Sentence	MLE Sentence	ROUGE Sentence
1	a green train traveling through a city on railroad tracks .	a train is on a track near some trees	a train is on a train in the .
2	two giraffes eating from a basket on a pole .	two giraffes standing in a zoo enclosure together .	two giraffes standing in a in a .
3	a baseball player reaching for a ball on the ground .	a baseball player is a bat field with a	a baseball player on a field with .
4	a man holding up a microphone talking .	a man is holding a in a the .	a man is holding a in a .

ROUGE Training Perf. (2,500 Ep Avg.)



CIDEr

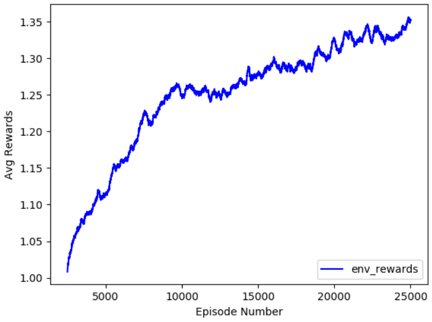
CIDEr is described in section 2.2 and is extremely computationally expensive as it compares cosine distances across n-grams between sentences. For this reason, we were only able to pretrain the critic for 25k episodes as it took ~18 hours (compared with 125k episodes over ~3 hours for BLEU) and train the model for 25k episodes which took another ~18 hours. The performance is not directly comparable although we consider the roughly 60x slower training time a limitation of the metric as a reward function. As the model did not train for as many episodes, the effects are more muted.

CIDER performed fairly reasonably in cases where it produced completed sentences although the large number of sentence fragments such as “a man holding a a on a a .” suggests the model is not sufficiently trained. In cases where CIDER outperformed MLE, it produces sentences which are more general somewhat comparable to the BLEU-2 model and the length penalty model discussed later.

Example Generated Sentences

#	Input Sentence	MLE Sentence	CIDER Sentence
1	a man is smiling by some wine barrels .	a man is smiling while holding a wine	a man holding a a on a a .
2	a piece of meat in a plastic container with broccoli .	a piece of broccoli and a white plate	a plate of a a a a .
3	a person sitting on a bench over looking the ocean .	a person sitting on a bench near the water .	a person that is sitting on a bench .
4	an elephant walks alone through some bushy grassland	a large elephant is walking in the in the .	a large elephant that is on a field .

CIDER Training Perf. (2,500 Ep Avg.)



PARA & PARA-F

PARA

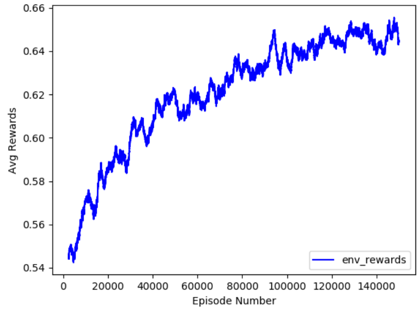
Comparing the cosine distance between BERT sentence embeddings was used as a proxy for semantic relatedness between sentences. The intention is the closer the BERT embeddings, the more similar the sentences and therefore the more likely they are a correct paraphrase. The main flaw with this approach is that fluency is not considered explicitly, and the generator attempts to find vulnerabilities in the BERT model to increase relatedness. The outcome is that while BLEU1 and ROUGE fall considerably, PARA, fluency and ESIM all increase versus the supervised model resulting in higher average performance.

The sentence quality is still not satisfactory qualitatively with the model repeating words and not understanding the connection between entities resulting in sentences which defy common sense such as “a man riding a surfboard on a surfboard”. Despite the model lacking an understanding of relationships, the fluency scores are still improved relative to the MLE model.

Example Generated Sentences

#	Input Sentence	MLE Sentence	PARA Sentence
1	a surfer is in the middle of an ocean wave	a man is on a surfboard in the ocean .	a man riding a surfboard on a surfboard
2	a single sheep grazes in a field of grass	a large sheep is standing in a field	a sheep with a sheep in a field
3	a wooden table with four different chairs sitting around it .	a wooden desk is shown with a chair and	a kitchen with a table chairs and chairs
4	a group of boats that are sitting in the water .	a group of boats in the the water the water .	a group of boats on a boat on the water

PARA Training Perf. (2,500 Ep Avg.)



PARA-F: The Challenges in Developing a Paraphrase Reward Function - Just Put it On the Table

One approach which was implemented to resolve the challenges PARA faces in understanding relationships between words was to include a fluency language model which would consider a sentence such as “a group of boats on a boat on the water” as low probability. This was our attempt at explicitly modelling the two conditions a generated sentence must satisfy to be considered a paraphrase: semantic similarity (BERT cosine distance) and fluency (normalized GPT perplexity).

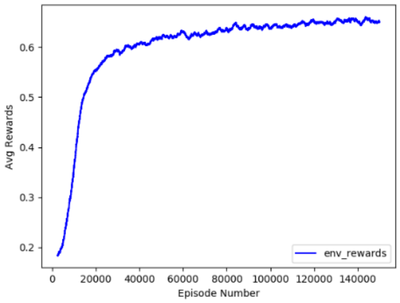
The fluency score increases dramatically to the highest score out of any configuration although the sentences lack semantic meaning and the model strongly prioritizes fluency over semantic similarity. The main issue the model faced was not sufficiently capturing semantic similarity with entities most entities being placed "in the middle" of something and being moved to "on a table" or "side of the road". This persisted for every model and weight combination we tried including using InferSent instead of BERT, normalizing the metrics before combining them, and changing the weights used to combine the scores. Multi objective reinforcement learning is an open research question and is very challenging in this case.

The advantage of this model is that the generated sentences tend to be syntactically correct although somewhat generic as much of the detail is removed.

Example Generated Sentences

#	Input Sentence	MLE Sentence	PARA-F Sentence
1	two zebras standing in grassy area facing toward one another .	two zebras standing in a grassy field near some trees	two zebras in the middle of the field .
2	a kitchen with a stainless steel stove .and white cabinets .	a kitchen with kitchen counter and cabinets	a kitchen in the middle of the kitchen .
3	some little boats on a lake with the sun shining .	a couple of people riding on a the .	two boats in the middle of the ocean .
4	two women are on the beach flying a kite .	two women are flying the beach with their kites .	two women in the middle of the beach .

PARA-F Training Perf. (2,500 Ep Avg.)



PARASIM & PARASIM-F

PARASIM

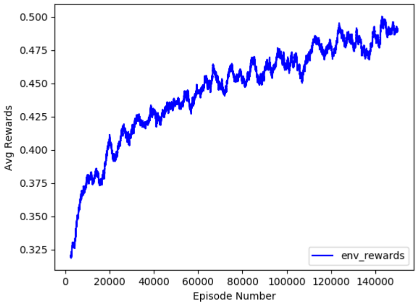
To further improve the focus on semantic relatedness we incorporate ESIM and evenly weight it with PARA. Part of the motivation is to capture the notion of using an ensemble to capture the different features of semantic relatedness. The inclusion of ESIM improved performance slightly in terms of fluency and ESIM although qualitatively the sentences are much worse, and it is definitely not worth the incremental training time as ESIM is a fairly expensive model to run both in terms of required computation and time.

The generated sentences still suffer from the same repeating words and misunderstanding of relationships between entities.

Example Generated Sentences

#	Input Sentence	MLE Sentence	PARASIM Sentence
1	a purple bus drives onto a city street .	a bus is in the middle of the road .	a bus bus on a street on a street street
2	small plane on tarmac with pilot at controls at airport .	a small jet is sitting on a runway .	a plane on a jet plane on a runway runway
3	a close up of a pizza on a tortilla	a pizza is on a table on a table	a pizza pizza with pizza on a pizza table
4	a young man sitting down talking on a phone .	a man is sitting on a cell phone	a man sitting on a cell phone with cell phone

PARASIM Training Perf. (2,500 Ep Avg.)



PARASIM-F

Similar to PARA, the lack of explicit fluency model was thought to be an architectural limitation of the approach. Therefore, similar to PARA-F we add fluency score to the average meaning it is weighted 2/3 semantic relatedness and 1/3 fluency. This is an oversimplification as ESIM also considers fluency at the extremes.

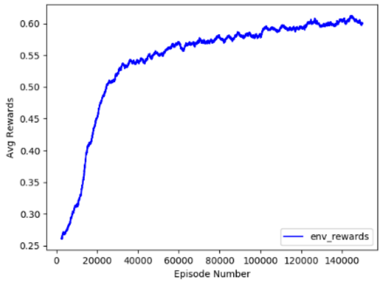
Similar to PARA-F the fluency score increases dramatically at the expense of sentences which are more generic and arguably lower in semantic relatedness. The agent discovers a brilliant strategy of generating somewhat templated sentences such as “a <object of the sentence> in the middle of the <scene location>”. The agent also follows a similar strategy to PARA-F of repeating words to maximize the ESIM and PARA scores e.g. “a woman on the tennis court on the court .”

As with PARASIM, the addition of ESIM to the reward function increases the overall score and sentence performance although the incremental training time is not worth the improvement in performance.

Example Generated Sentences

#	Input Sentence	MLE Sentence	PARASIM-F Sentence
1	a player readies for a swing during a tennis game	a tennis player is holding a racket	a tennis player on the court on the court .
2	a dog is herding sheep in a field .	a dog is standing in a field with a	a dog in the middle of the field .
3	there are several kites being flown by the water	a large kite flying kites in the water .	a kite in the middle of the sky .
4	a brown and white colt stands in a fenced area .	a large white of in a a in of	a brown bear in the middle of the field .

PARASIM-F Training Perf. (2,500 Ep Avg.)



ESIM

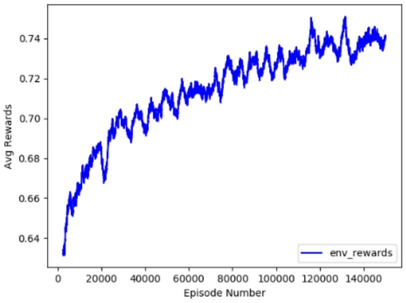
As discussed in section 2.2, ESIM and other sentence pair models which model the local interactions tend to perform better on paraphrase identification than sentence encoding models. Training the RL model to maximize the probability, ESIM predicts the generated sentence is a paraphrase in a similar approach to PARA although as ESIM has been trained on sentences which are not fluent, it is also able to capture some of this objective.

Optimizing ESIM achieves performance inline with PARA and PARASIM although the qualitative performance is worse with sentence fragments and repeated words.

Example Generated Sentences

#	Input Sentence	MLE Sentence	ESIM Sentence
1	a vase with yellow roses in it on a table	a vase filled with yellow flowers on a table .	a vase of flowers in a vase in a vase
2	a blurred photo of a person traveling on a motorcycle .	a person riding a motorcycle down a road .	a motorcycle is riding a motorcycle in the street
3	a toasted sandwich and french fries on a plate .	a plate of food on a on on .	a sandwich sandwich on a plate on a table
4	there are many snowboarders on the snowy mountain	a group of people on snow skis on the .	a group of people skiing skiing in the snow

ESIM Training Perf. (2,500 Ep Avg.)



Adversarial Approach

Given the difficulties in designing a reward function which adequately captures the objective, the final approach is to use adversarial training. In adversarial training, the generator is still trying to maximize the probability the ESIM model believes the generated sentence is a paraphrase, although every 6000 iterations, the discriminator model relearns based on its predictions. This results in the generator trying to fool the discriminator and each learning from the other model’s output.

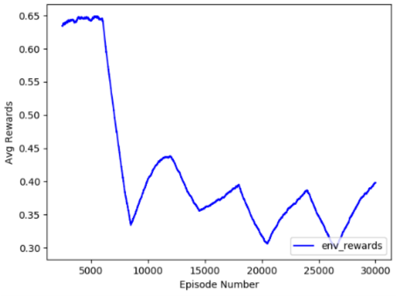
This resulted in the highest average score and qualitative output. Although the coding is more elaborate, it is a far more elegant solution than manually devising a reward function iteratively. Some sentences still contain grammatical errors as fluency is not explicitly optimized, although the model performs quite a bit better than the supervised model. Even using MLE the adversarial model outperforms the supervised model using beam decoding.

When examining the training performance graph, it is clear when the discriminator model improves as the generator performance decreases sharply then continues to improve based on the reward feedback from the new model.

Example Generated Sentences

#	Input Sentence	MLE Sentence	Adversarial Sentence
1	vintage image of a group of suitcases sitting on a sidewalk	three suitcases are sitting on a sidewalk on a sidewalk .	three suitcases are sitting on a sidewalk .
2	a red motorcycle that is going down the road	a motorcycle is on the road near a road .	a motorcycle is in the middle of the road .
3	two jets are flying against a bright blue sky .	a pair of airplanes flying in the sky .	two airplanes flying in the sky above some trees .
4	a group of people relaxing on the beach	a group of people on the beach with umbrellas	three people are walking on the beach .

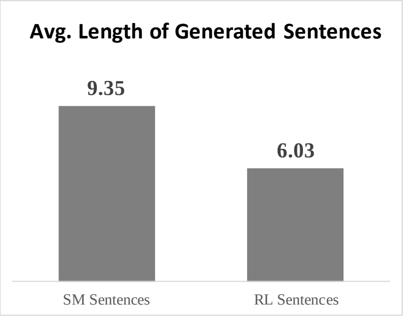
Adv. Training Perf. (2,500 Ep Avg.)



Auxiliary Reward Functions | Length Penalty

The largest advantage of using RL instead of supervised learning for NLG is that you are able to specify an arbitrary reward function. This means that if you can capture a linguistic style or feature in a metric, then you can optimize the generated language towards it. To demonstrate the power of this approach we generate sentences which are penalized for length while still using the adversarial objective. This means the generator is trying to generate sentences which can fool the discriminator but also be short enough to satisfy the length objective.

The outcome is very impressive with short coherent sentences similar to a summarization objective. The model "runs out of space" in many examples where it ends in a sentence fragment e.g. "a truck is parked on" although this still demonstrates the proof of concept.



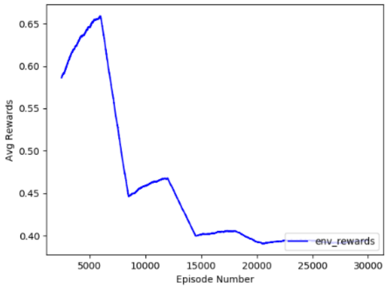
Combining the reward functions is a very hard problem and while different auxiliary metrics were implemented and tested, none were as successful as the length penalty therefore it has been left to future work.

When examining the training performance, the generator performance does not improve after ~15,000 episodes as the discriminator has learned that shorter sentences are much more likely to be “fake” sentences. This means the ability of the generator to fool the discriminator is heavily handicapped.

Example Generated Sentences

#	Input Sentence	MLE Sentence	Adv. Short Sentence
1	different styles of wine glasses lined up on the table	a glass of wine glasses on a table .	a table with glasses of wine
2	two small birds perched on tree branches together	two birds sitting on a tree branch with a .	two birds sitting on a branch
3	the man is skateboarding down the road with his dog .	a man is walking down with a dog .	a man is walking down the
4	a couple of giraffe standing next to each other .	two giraffes standing in a grassy field with trees	two giraffes standing in a field

Length Training Perf. (2,500 Ep Avg.)



Impact of Using MCTS

We test using MCTS for only the adversarial model as it is very computationally expensive. To get a sense for the impact of using MCTS, we use a random sample of 800 test sentences and only run 100 simulations per step as this still took over 12 hours to run.

Test Set Performance

Model Name	BLEU-1	BLEU-2	ROUGE	PARA	Fluency	ESIM	Average
Adversarial	0.3727	0.1577	0.3646	0.6692	0.8507	0.7409	0.5260
MCTS-100	0.3603	0.1475	0.3490	0.6641	0.8327	0.7392	0.5155
Impact on Perf.	-3.3%	-6.5%	-4.3%	-0.8%	-2.1%	-0.2%	-2.0%

The results are surprisingly weak with MCTS reducing performance for ParaPhrasee versus simply using the adversarial model. While the drop in performance is not very substantial, it is likely due to insufficient simulations per step as AlphaGo used 1,600 simulations per step. In contrast with FrozenLake, applying MCTS to ParaPhrasee was a very challenging coding problem. Although we spent a large amount of time thoroughly writing and evaluating the code, it is still possible there is an implementation error. Python is too high level a language to implement MCTS efficiently which makes thorough experimentation intractable given our access to resources.

Summary of Key Findings from Reinforcement Learning Paraphrase Generation

Based on the experiments on the ParaPhrasee environment, the following findings can be concluded:

- BLEU1 performs poorly as a reward function as the generated sentences are mostly incoherent and are composed of connecting words such as “in the a”.
- BLEU2 performs surprisingly well as reward function in terms of generating fluent sentences although they often lack semantic similarity.
- ROUGE generates sentences which perform better than BLEU1 although with similar structural weaknesses.
- CIDEr is incredibly resource intensive to train and although we were unable to train it for the same duration as the other models; the results were subpar with lots of poorly formed sentences and sentences which lacked semantic similarity.
- PARA & PARASIM perform similarly with ESIM slightly improving performance although both models generating sentences which misunderstand the relationships between entities and repeat words.
- ESIM performs similarly to PARA & PARASIM.
- PARA_F & PARASIM_F perform similarly with the explicit addition of fluency improving the coherence of generated sentences at the expense of specificity and with sentences often following the pattern of objects being placed “in the middle” of something.
- Adversarial training achieves the best performance across metrics improving over the supervised model and generating fluent sentences.
- Adding a length penalty as an auxiliary metric causes the model to produce shorter sentences which are often equally fluent at the expense of sentence detail.
- Designing a reward function is extremely challenging with very minor changes to structure often resulting in significant changes to performance.
    - Capturing the intended performance of poorly defined concepts such as “paraphrase quality” in a simple evaluation metric is an extremely challenging problem and relying on adversarial training appears to be the most principled and best performing approach.

5 Discussion

This chapter revisits the initial objectives defined in the project proposal and the beginning of the project against the achievements resulting from the research which was undertaken. The research is then reviewed in the wider context of the prior work discussed in the Context section and the generalizability and validity of the results are considered. Finally, we discuss the implications of the results and the recommendations which follow.

5.1 Objectives Fulfilment

Summary of Answers to Research Questions

1. *What is the most effective reinforcement learning reward function for paraphrase generation?*

   As discussed in section 4.3, we suggest using adversarial learning approaches in situations where reward functions are very challenging if not impossible to design such as paraphrase generation.

2. *What is the best sequence to sequence architecture for paraphrase generation?*

   As discussed in section 4.1, the definition of "best" depends on the user's objectives as pretrained encoders such as BERT achieve slightly higher results on the automatic metrics. However, this is at the cost of higher training time. Depending on the size of the training set and the sentence length, training a GRU model from random initialization will likely result in comparable performance. It should also be noted that BERT was not fine-tuned for performance on this task which is an integral part of achieving state-of-the-art performance.

3. *How can knowledge obtained through supervised learning be leveraged to decrease computation requirements and training time for RL agents?*

   As discussed in section 4.2, there are many approaches to transferring knowledge between neural models. Policy distillation, while elegant, is very fragile in a RL context as there are many interrelated hyperparameters which cause significant variations in performance. Simply transferring the weights between similar architectures and pretraining the critic model in an actor-critic architecture resulted in most stable and achieved the best performance.

4. *How does performance vary between maximum likelihood estimation supervised training and reinforcement learning objectives?*

As discussed in section 4.3, using reinforcement learning for paraphrase generation improves performance on metrics related to the reward function which can have unforeseen impacts on generation quality. Supervised training performs reasonably although fine-tuning using RL improves performance when using the right reward function at the expense of increased computation. Another benefit as seen with adding a length penalty to the adversarial reward function is the agent is able to learn functions without explicit data as would be required with supervised learning.

5. *What is the impact of using Monte Carlo Tree Search on performance for trained reinforcement learning models?*

   As discussed in section 4.2, Monte Carlo Tree Search (MCTS) improves the performance of the RL agent through simulating different actions and determining which will result in the best performance. While the performance of FrozenLake improved through the use of MCTS on the final model, it decreased slightly when applied to ParaPhrasee. As discussed in 4.3 further experimentation is required to understand why MCTS does not improve the performance of ParaPhrasee.

Comparison to Objectives at Outset

In the table below we consider the initial objective alongside the testable result and the outcome.

Objective	Testable Result	Outcome
Thorough literature review of approaches to natural language generation, particularly paraphrase generation Thorough literature review of existing automatic evaluation methods	Comprehensive "Context" section in final project report	Completed thorough literature review covering: both traditional and neural approaches to NLG, automatic evaluation approaches and the key metrics used, deep reinforcement learning, and approaches for paraphrase generation
Evaluation of performance of different encoder models on supervised paraphrase generation	Results comparing performance across numerous metrics	Coded and evaluated main encoder models and tested across different validation loss values
Trained GRU model on paraphrase sentence pairs	Trained paraphrase generation model using a specified encoder and decoder model	Trained multiple supervised paraphrase generation models using GRU decoder
Fine-tuned RL language model / decoder on specified objective (reward function)	Trained RL paraphrase generation model fine-tuned model on reward resulting in improved performance	Successfully developed approach to transfer knowledge from the supervised model to a reinforcement learning model an improve performance of an arbitrary reward function.

Identify best existing metric for evaluating performance in paraphrase generation	Use of principled metric with theoretical justification to assess model performance	Used 6 metrics to evaluate model performance alongside qualitative sentence performance. Determined adversarial reward functions perform best for generation performance.
Contribute to "world's body of knowledge" (Dawson, 2009, p. 17)	Research demonstrating the best approach in applying reinforcement learning to fine-tune paraphrase generation models	Thoroughly demonstrated approaches which work and those which don't across supervised learning generation, two toy RL environments and applying RL to a complex problem.
Develop GUI / tool Phrasee can use to generate sentential paraphrases	Tool in which short form text is entered and optimized text is returned with the same semantic meaning	Developed command line tool which takes an arbitrary input sentence and generates a sentential paraphrase with the selected model.
Develop list of future projects and extensions which build off this work	Comprehensive 'Future Works' section in final project report	Suggested multiple projects in order of deviation from current work. Proved concept of controllable generation using multi-objective reinforcement learning

5.2 Research in a Wider Perspective

The results achieved through using reinforcement learning to generate paraphrases are best compared with Li et al. (2018) and Gupta et al. (2018) who both used neural models for generating paraphrases. Gupta et al. achieved BLEU performance of ~41 on MS-COCO which compares to our performance of ~40 when optimizing for BLEU-1 directly using RL. It should be noted that our results are achieved using only a sample of the data, so the results are not directly comparable although are indicative. Gupta et al. also include the BLEU performance of Prakash et al.'s (2016) Residual LSTM model of ~37. Given this it is evident our model performs comparably with state-of-the-art approaches in terms of BLEU on MS-COCO.

Li et al. (2018) did not evaluate using MS-COCO and instead used Quora and Twitter. One advantage of our approach over theirs is adversarial learning is more efficient than the inverse reinforcement learning approach they used. As they did not report their training time, we are unable to compare although we suspect that our approach is more computationally efficient. Although it is on a different dataset, the generated sentences included in their paper are fairly poor quality. One example is: "why is donald trump still ducking his income tax return issue" paraphrasing to "Why did trump deal tax issue?".

The approach developed for using reinforcement learning in a large state and action space can be applied to other NLP problems or RL problems in general. The ability to optimize for an arbitrary auxiliary function alongside the adversarial approach allows for applications into controllable generation.

5.3 Results Validity and Generalizability

The results are most transferrable to other short-form generation tasks such as short-text summarization, caption generation, translation, and most dialogue systems. However, the results achieved in this work cannot be generalized to all form of text as language is incredibly diverse. One of the biggest limitations to the approach is in generation beyond short-form text such as stories or articles. Common approaches to generating longer form text are to rely on hierarchical models similar to the approaches used in classical NLG where different models were responsible for different aspects of the generation e.g. planning vs realization.

Another limitation is the specific use of language examined in this project is fairly structured relative to general language. The language used to describe key elements of a scene is a subset of general language and as such the agents are able to develop strategies for generation such as an <object> is <in relation> to a <scene> and achieve fairly high accuracy. These templated style techniques do not scale to the general problem of generation for example in dialogue systems. Initial experiments using Twitter and WikiAnswers data suffered from poor performance due to the large amount of noise in each of the datasets from poor grammar and the sentence fragments which occurred naturally in the datasets. When testing input sentences that are very different from the training data, the results are poor and unpredictable.

The principles developed with the intermediary RL environments is likely generalizable to the broader problem of reinforcement learning. The more general problem of solving RL in massive state and action spaces with limited computing resources using supervised warm-starting is definitely generalizable. The obvious limitation of this approach being the reduction in exploration may reduce long-term performance.

5.4 Recommendations

Having spent a considerable amount of time researching prior work and experimenting with both supervised and reinforcement learning approaches to NLP problems, the main recommendation is

RL is best left to situations where supervised learning cannot be applied. The approaches and algorithms are far more mature in supervised learning and the error signal is much stronger which results in much faster and more stable training. RL in contrast, is quite mature in certain narrow problems such as path searching and low dimensional action spaces. However, there are many open problems which are fundamental to the field. As is commonly joked about in the Deep RL community, it is the only area of machine learning where it is socially acceptable to train on the test set.

If the application requires state-of-the-art performance and can be structured as a RL problem (with an emphasis on optimizing sequential decisions), then RL is likely to improve the performance. The main challenge is defining the reward function which is the key theme throughout this paper. If the user has a metric which captures the problem elegantly (e.g. winning or losing a game) then RL is likely to substantially outperform MLE approaches to optimization. Where this cannot be defined, we suggest using an adversarial agent analogous to the use of self-play in RL for games. While instability and large amounts of computation can become problematic, RL proposes a set of techniques which are likely to unlock the next wave of performance in certain tasks within NLP.

6 Evaluation, Reflections, and Conclusions

This chapter reviews the project as a whole including the choice of objectives, literature reviewed, methods applied, and planning. The main conclusions and contributions of the work are highlighted, and their implications are discussed. The reflections section discusses learnings from the project overall and the future works section discusses potential extensions to the work and other related interesting research questions.

6.1 Evaluation

The original objectives were well formed in terms of being ambitious although still attainable. The time spent structuring the project, learning about NLP, and completing a thorough literature review were the cornerstone of the project success. Having the ability to discuss ideas with my supervisor Ed, and the data science team at Phrasee Neil and Elena, helped significantly in achieving the objectives.

The literature reviewed was very interesting as it covered a wide variety of research areas which approached similar problems using different techniques. It is interesting to see how long automatic evaluation for NLP has been an open question with relatively weak progress. BLEU was released in 2002 and is still the dominant automatic measure despite its well acknowledged weaknesses. In contrast, the reinforcement learning algorithms and approaches were well ahead of their time with solutions to problems which required improvements in computing hardware to unlock their potential. NLG overall is somewhere in the middle of these two with significant improvements particularly in short form text resulting from the success of neural architectures including large deep transformers. However, approaches for controllable generation and longer text are still weak.

While the project was successful in achieving its objectives, it is clear that the current state-of-the-art in NLG is still not ready for most production applications. Phrasee requires generating subject lines which are high performing but also grammatically flawless and fluent. While the agent is generally able to generate lines that are convincing, it often makes mistakes which would not be tolerable in Phrasee’s use case without human editing or further checks. Therefore, we are considering applying it as a preliminary tool which humans must approve before it is sent.

Developing the intermediary environments was very useful in isolating the components of the configuration which contributed most to performance. The instability in the system meant that a well understood environment which was easier to compute was integral to running many experiments.

The main contributions of the work are a tool Phrasee can use to generate paraphrases for subject lines for their clients, a generalizable approach to applying RL for NLP tasks with large state and action spaces, and identifying adversarial generation as the best reward metric for fine-tuning paraphrase generation using reinforcement learning.

6.2 Reflections

One of the key reflections when working with RL agents is how temperamental they are even in simple environments. The amount of uncertainty resulting from a random initialization in model weights for both the actor, the critic, the encoder, and the environment makes learning from scratch very unstable. Early experiments on the simplest environment tested (CartPole), revealed that when training all components from scratch, one of the most important variables to determining whether the configuration would succeed is the random seed used. This makes experimentation very challenging as each experiment must then be run using different random seeds to reduce random noise.

Another interesting reflection is how long agents take to train which is often omitted from key RL papers. Monte Carlo Tree Search in particular is a very impressive approach theoretically although David Silver's (2016) observation that supervised learning achieved comparable results with 15,000x less computation was very much noted.

When considering existing automatic evaluation metrics, it quickly becomes very evident how weakly they reflect human judgment and how primitive the dominant techniques are. This is not for a lack of trying within the research community as discussed in section 2.2 although is a testament to how complicated the properties of language are to quantify. Two additional problems which are not often mentioned as criteria in the automatic evaluation literature examined is the speed of computing the metric and to a lesser extent its simplicity. We attempted to train two RL agents to optimize for METEOR and CIDEr and while we were able to train a version on CIDEr, the significantly greater training time (60x+) made training the models at scale intractable. It seems likely that the future of evaluation metrics will be neural given its obvious advantages. We surmise that it is likely task specific evaluation models will emerge given the wide variety in NLP tasks.

A very similar observation of the challenges of capturing language in a metric was discovered when trying to design a RL reward function which would produce high performing paraphrases. The model would continuously find strategies which “missed the point” of the task and solve the problem in the narrowest way possible. A particularly frustrating example was when trying to combine semantic similarity and fluency. The model would attempt to put everything “on a table” as this would improve fluency just enough to overcome the loss in semantic similarity. If you then refine the loss function to make semantic similarity more important then word and pattern repetition would become very common (e.g. a man is in a park in a park).

6.3 Future Work

Given the limitations in timing for this project, there were many more ideas which we wanted to implement although were unable. The future work is ordered from most immediate to higher level.

- Hyperparameter tuning and neural architecture search: given the long compute times we were unable to test multiple neural architectures or hyperparameter settings. There was a substantial difference in performance between using Adam and SGD for the RL model therefore it stands to reason that there are other hyperparameter settings and neural architectures which would improve performance.
- Handling out-of-vocabulary words and named entities: for Phrasee and other industrial applications, handling the vocabulary and ensuring specific named entities are in the generated sentence is an extremely important problem. It is common to convert infrequent words to UNK tokens although doing so in an intelligent way would improve performance.
- End-to-end finetuning: we set up the problem as the encoder is static and the decoder is what is fine-tuned. Another formulation of the problem which could improve performance is to allow the agent to train both the encoder and the decoder using RL.
- Using attention for the RL decoder: similarly, we do not use attention in the decoder which could improve performance.
- Improving the efficiency of knowledge transfer between the supervised model and RL model: we explored using policy distillation although the results were very fragile resulting in a significant deterioration in performance if not set correctly. Using imitation learning approaches would likely also be a good area to explore.

- Using auxiliary metrics for controllable generation: as demonstrated with the success in generating sentences which are short but also good quality paraphrases, it is possible to include auxiliary metrics to do controllable generation.

6.4 Conclusions

The project was extremely interesting and covered a wide variety of problems which are fundamental to both NLP and RL. Understanding paraphrases is a key problem in natural language understanding and language in general. Most RL papers are restricted to solving problems with small action spaces and the proposed approaches do not scale to large action spaces as the space is too large to explore efficiently. We propose an approach to efficiently generate paraphrases that are somewhat controllable using an arbitrary reward function.

Applying RL to NLP is still relatively new and will lead to an explosion across different NLP tasks as the techniques are better implemented and best practices are developed. Projects building off the future works section are likely to produce interesting results and the next area of focus for us is controllable generation.

7 Glossary

DRL – Deep Reinforcement Learning: the application of (deep) neural networks to the reinforcement learning problem in approximating the environment’s state, value, or reward function.

GANs – Generative Adversarial Networks: a framework popularized by Goodfellow et al. (2014) for generating images through training two models which compete to generate and discriminate the generated images respectively.

GRU – Gated Recurrent Unit Network: an RNN architecture designed to overcome the vanishing gradient problem using memory gates with fewer parameters than the LSTM architecture.

LSTM – Long Short-Term Memory Network: an architecture designed to overcome the vanishing gradient problem using memory gates to model long-term dependencies.

MCTS – Monte Carlo Tree Search: a search algorithm popularized in model-based reinforcement learning for games. Builds a tree of the possible moves through rollouts and selects the move with the maximum expected value.

MLE – Maximum Likelihood Estimation: an approach for estimating the parameters of a model through maximizing the likelihood the observed data occurs.

NLP – Natural Language Processing: applying computational techniques to the problem of natural language (including speech).

NLU – Natural Language Understanding: the application of computational techniques to reading comprehension of text.

RL – Reinforcement Learning: a class of approaches to find actions which maximize the total reward an agent receives as it interacts with its environment (Sutton and Barto, 1998).

RNN – Recurrent Neural Network: a class of neural networks used to model sequential data through using the hidden layer alongside the input to predict the next time step.

8 Appendix A – References

1. Agirre, E., Cer, D., Diab, M., Lopez-Gazpio, I. and Specia, L., 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

2. Alonso, E. and Mondragón, E. 2019. INM426 Software Agents Class Notes.

3. Anderson, P., Fernando, B., Johnson, M. and Gould, S., 2016, October. Spice: Semantic propositional image caption evaluation. In European Conference on Computer Vision (pp. 382-398). Springer, Cham.

4. Arora, S., Liang, Y. and Ma, T., 2016. A simple but tough-to-beat baseline for sentence embeddings.

5. Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A., 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

6. Bahdanau, D., Cho, K. and Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

7. Bannard, C. and Callison-Burch, C., 2005, June. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 597-604). Association for Computational Linguistics.

8. Barto, A.G., Sutton, R.S. and Anderson, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), pp.834-846.

9. Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C., 2003. A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), pp.1137-1155.

10. Bhagat, R. and Hovy, E., 2013. What is a paraphrase?. *Computational Linguistics*, *39*(3), pp.463-472.

11. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S., 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), pp.1-43.

12. Brunskill, E. 2019. “Given a Model of the World” Stanford, CS234: Reinforcement Learning Lecture 2. Available at: https://www.youtube.com/watch?v=E3f2Camj0Is

13. Callison-Burch, C., Cohn, T. and Lapata, M., 2008, August. Parametric: An automatic evaluation metric for paraphrasing. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)* (pp. 97-104).

14. Cao, Z., Luo, C., Li, W. and Li, S., 2017, February. Joint copying and restricted generation for paraphrase. In *Thirty-First AAAI Conference on Artificial Intelligence*.

15. Che, T., Li, Y., Zhang, R., Hjelm, R.D., Li, W., Song, Y. and Bengio, Y., 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

16. 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*.

17. Cheng, Y., Wang, D., Zhou, P. and Zhang, T., 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

18. Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A., 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

19. Cui, Y., Yang, G., Veit, A., Huang, X. and Belongie, S., 2018. Learning to evaluate image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5804-5812).

20. Culicover, P.W., 1968. Paraphrase generation and information retrieval from stored text. Mech. Translat. & Comp. Linguistics, 11(3-4), pp.78-88.

21. Dawson, C. 2009, Introduction to research methods: a practical guide for anyone undertaking a research project, 4th edn, How To Books, Oxford.

22. DeepMind (2019) “AlphaStar: Mastering the Real-Time Strategy Game StarCraft II” Available at: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ (Accessed: 26 April 2019).

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

24. Doddington, G., 2002, March. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research* (pp. 138-145). Morgan Kaufmann Publishers Inc..

25. Dolan, W.B. and Brockett, C., 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

26. Dong, L., Mallinson, J., Reddy, S. and Lapata, M., 2017. Learning to paraphrase for question answering. arXiv preprint arXiv:1708.06022.

27. Dras, M., 1999. *Tree adjoining grammar and the reluctant paraphrasing of text*. Sydney: Macquarie University.

28. Dušek, O., Novikova, J. and Rieser, V., 2018. Findings of the E2E NLG challenge. *arXiv preprint arXiv:1810.01170*.

29. Elliott, D. and Keller, F., 2014, June. Comparing automatic evaluation measures for image description. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 452-457).

30. Fader, A., Zettlemoyer, L. and Etzioni, O., 2013, August. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1608-1618).

31. Fomicheva, M., Bel, N., Specia, L., da Cunha, I. and Malinovskiy, A., 2016, August. CobaltF: a fluent metric for MT evaluation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers* (pp. 483-490).

32. Fujita, A. and Sato, S., 2008, August. A probabilistic model for measuring grammaticality and similarity of automatically generated paraphrases of predicate phrases. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1* (pp. 225-232). Association for Computational Linguistics.

33. Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S.M. and Vinyals, O., 2018. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*.

34. Gardent, C. and Kow, E., 2005. Generating and selecting grammatical paraphrases. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*.

35. Gardent, C., Amoia, M. and Jacquey, E., 2004, July. Paraphrastic grammars. In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation* (pp. 73-80). Association for Computational Linguistics.

36. Gatt, A. and Krahmer, E., 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, *61*, pp.65-170.

37. Gkatzia, D. and Mahamood, S., 2015, September. A snapshot of NLG evaluation practices 2005-2014. In *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)* (pp. 57-60).

38. Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.

39. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

40. Gupta, A., Agarwal, A., Singh, P. and Rai, P., 2018, April. A deep generative framework for paraphrase generation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

41. Han, L., 2016. Machine Translation Evaluation Resources and Methods: A Survey. *arXiv preprint arXiv:1605.04515*.

42. Hasselt, H. 2018. "Planning and Models" DeepMind, Advanced Deep Learning & Reinforcement Learning Lecture 14. Available at: https://www.youtube.com/watch?v=Xrxrd8nl4YI

43. He, H. and Lin, J., 2016, June. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 937-948).

44. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D., 2018, April. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

45. Hinton, G., Vinyals, O. and Dean, J., 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

46. Iyyer, M., Manjunatha, V., Boyd-Graber, J. and Daumé III, H., 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (Vol. 1, pp. 1681-1691).

47. James, S., Konidaris, G. and Rosman, B., 2017, February. An analysis of monte carlo tree search. In *Thirty-First AAAI Conference on Artificial Intelligence*.

48. Joshi, M., Chen, D., Liu, Y., Weld, D.S., Zettlemoyer, L. and Levy, O., 2019. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

49. Kaelbling, L.P., Littman, M.L. and Moore, A.W., 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, pp.237-285.

50. Kaggle (2017) “Quora Question Pairs” Available at: https://www.kaggle.com/c/quora-question-pairs (Accessed: 30 May 2019).

51. Kalchbrenner, N. and Blunsom, P., 2013, October. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1700-1709).

52. Kalchbrenner, N., Grefenstette, E. and Blunsom, P., 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

53. Kann, K., Rothe, S. and Filippova, K., 2018. Sentence-Level Fluency Evaluation: References Help, But Can Be Spared!. *arXiv preprint arXiv:1809.08731*.

54. Keskar, N.S. and Socher, R., 2017. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*.

55. Kim, Y., Denton, C., Hoang, L. and Rush, A.M., 2017. Structured attention networks. *arXiv preprint arXiv:1702.00887*.

56. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

57. Kiros, R., Zhu, Y., Salakhutdinov, R.R., Zemel, R., Urtasun, R., Torralba, A. and Fidler, S., 2015. Skip-thought vectors. In *Advances in neural information processing systems* (pp. 3294-3302).

58. Kolesnyk, V., Rocktäschel, T. and Riedel, S., 2016. Generating natural language inference chains. *arXiv preprint arXiv:1606.01404*.

59. Kondadadi, R., Howald, B. and Schilder, F., 2013, August. A statistical nlg framework for aggregated planning and realization. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 1406-1415).

60. Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

61. Kukich K. (1987) Where do Phrases Come from: Some Preliminary Experiments in Connectionist Phrase Generation. In: Kempen G. (eds) Natural Language Generation. NATO ASI Series (Series E: Applied Sciences), vol 135. Springer, Dordrecht

62. Kusner, M., Sun, Y., Kolkin, N. and Weinberger, K., 2015, June. From word embeddings to document distances. In *International conference on machine learning* (pp. 957-966).

63. Lan, W. and Xu, W., 2018, August. Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 3890-3902).

64. Lan, W., Qiu, S., He, H. and Xu, W., 2017. A continuously growing dataset of sentential paraphrases. *arXiv preprint arXiv:1708.00391*.

65. Lavie, A. and Agarwal, A., 2007, June. METEOR: An automatic metric for MT evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation* (pp. 228-231). Association for Computational Linguistics.

66. Le, Q. and Mikolov, T., 2014, January. Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188-1196).

67. Lemon, O., 2008, June. Adaptive natural language generation in dialogue using reinforcement learning. In *Proceedings of the 12th SEMdial Workshop on the Semantics and Pragmatics of Dialogues* (pp. 149-156).

68. Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J. and Jurafsky, D., 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

69. Li, Y., 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

70. Li, Z., Jiang, X., Shang, L. and Li, H., 2017. Paraphrase generation with deep reinforcement learning. *arXiv preprint arXiv:1711.00279*.

71. Li, Z., Jiang, X., Shang, L. and Liu, Q., 2019. Decomposable Neural Paraphrase Generation. arXiv preprint arXiv:1906.09741.

72. Lin, C.Y., 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out* (pp. 74-81).

73. Lin, K., Li, D., He, X., Zhang, Z. and Sun, M.T., 2017. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems* (pp. 3155-3165).

74. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.

75. Lin, Y., Tan, Y.C. and Frank, R., 2019. Open Sesame: Getting Inside BERT's Linguistic Knowledge. *arXiv preprint arXiv:1906.01698*.

76. Linzen, T., Dupoux, E. and Goldberg, Y., 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, *4*, pp.521-535.

77. Liu, S., Zhu, Z., Ye, N., Guadarrama, S. and Murphy, K., 2017. Improved image captioning via policy gradient optimization of spider. In Proceedings of the IEEE international conference on computer vision (pp. 873-881).

78. Liu, S., Zhu, Z., Ye, N., Guadarrama, S. and Murphy, K., 2017. Improved image captioning via policy gradient optimization of spider. In *Proceedings of the IEEE international conference on computer vision* (pp. 873-881).

79. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

80. Madnani, N. and Dorr, B.J., 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, *36*(3), pp.341-387.

81. Martindale, M.J. and Carpuat, M., 2018. Fluency over adequacy: A pilot study in measuring user trust in imperfect MT. *arXiv preprint arXiv:1802.06041*.

82. McDonald, D.D., 2010. Natural Language Generation. Handbook of Natural Language Processing, 2, pp.121-144.

83. McKeown, K.R., 1980. Paraphrasing using given and new information in a question-answer system. *Technical Reports (CIS)*, p.723.

84. Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

85. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937).

86. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

87. Montone, G., O'Regan, J.K. and Terekhov, A.V., 2017. Gradual Tuning: a better way of Fine Tuning the parameters of a Deep Neural Network. *arXiv preprint arXiv:1711.10177*.

88. Mutton, A., Dras, M., Wan, S. and Dale, R., 2007, June. GLEU: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics* (pp. 344-351).

89. Napoles, C., Sakaguchi, K. and Tetreault, J., 2016. There's No Comparison: Reference-less Evaluation Metrics in Grammatical Error Correction. *arXiv preprint arXiv:1610.02124*.

90. Neubig, G. 2019. “Reinforcement Learning” Carnegie Mellon University, CS 11-747, Neural Networks for NLP. Available at: https://www.youtube.com/watch?v=lYXhT9JvB98&t=355s

91. Novikova, J., Dušek, O., Curry, A.C. and Rieser, V., 2017. Why we need new evaluation metrics for NLG. arXiv preprint arXiv:1707.06875.

92. Pan, S.J. and Yang, Q., 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), pp.1345-1359.

93. PapersWithCode, 2019. Language Modelling on Penn Treebank (Word Level). Available at: https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word (Accessed: 26 April 2019).

94. Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 311-318). Association for Computational Linguistics.

95. Parikh, A.P., Täckström, O., Das, D. and Uszkoreit, J., 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

96. Parisotto, E., Ba, J.L. and Salakhutdinov, R., 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

97. Paulus, R., Xiong, C. and Socher, R., 2017. A deep reinforced model for abstractive summarization. arXiv preprint arXiv:1705.04304.

98. Pennington, J., Socher, R. and Manning, C., 2014, October. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

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

100. Prakash, A., Hasan, S.A., Lee, K., Datla, V., Qadir, A., Liu, J. and Farri, O., 2016. Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098*.

101. Quirk, C., Brockett, C. and Dolan, W., 2004. Monolingual machine translation for paraphrase generation. In *Proceedings of the 2004 conference on empirical methods in natural language processing* (pp. 142-149).

102. Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I., 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

103. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I., 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8).

104. Ranzato, M.A., Chopra, S., Auli, M. and Zaremba, W., 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

105. Reiter, E. and Dale, R., 2000. *Building natural language generation systems*. Cambridge university press.

106. Rieser, V. and Lemon, O., 2009. Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical methods in natural language generation* (pp. 105-120). Springer, Berlin, Heidelberg.

107. Rus, V., Wyse, B., Piwek, P., Lintean, M., Stoyanchev, S. and Moldovan, C., 2011, September. Question generation shared task and evaluation challenge–status report. In Proceedings of the 13th European Workshop on Natural Language Generation (pp. 318-320).

108. Rusu, A.A., Colmenarejo, S.G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K. and Hadsell, R., 2015. Policy distillation. *arXiv preprint arXiv:1511.06295*.

109. Schmitt, S., Hudson, J.J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W.M., Leibo, J.Z., Kuttler, H., Zisserman, A., Simonyan, K. and Eslami, S.M., 2018. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*.

110. Shen, D., Wang, G., Wang, W., Min, M.R., Su, Q., Zhang, Y., Li, C., Henao, R. and Carin, L., 2018. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *arXiv preprint arXiv:1805.09843*.

111. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), p.484.

112. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of go without human knowledge. Nature, 550(7676), p.354.

113. Snover, M., Dorr, B., Schwartz, R., Micciulla, L. and Makhoul, J., 2006, August. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas* (Vol. 200, No. 6).

114. Socher, R., Pennington, J., Huang, E.H., Ng, A.Y. and Manning, C.D., 2011, July. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 151-161). Association for Computational Linguistics.

115. Stent, A., Marge, M. and Singhai, M., 2005, February. Evaluating evaluation methods for generation in the presence of variation. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 341-351). Springer, Berlin, Heidelberg.

116. Sutskever, I., Vinyals, O. and Le, Q.V., 2014. Sequence to sequence learning with neural networks. Advances in NIPS.

117. Sutton, R.S. and Barto, A.G., 1998. Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.

118. Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

119. Sutton, R.S., McAllester, D.A., Singh, S.P. and Mansour, Y., 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057-1063).

120. Tai, K.S., Socher, R. and Manning, C.D., 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

121. Taylor, M.E. and Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(Jul), pp.1633-1685.

122. Tesauro, G., 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), pp.58-68.

123. Toury, G., 2012. *Descriptive translation studies and beyond: Revised edition* (Vol. 100). John Benjamins Publishing.

124. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

125. Vedantam, R., Lawrence Zitnick, C. and Parikh, D., 2015. Cider: Consensus-based image description evaluation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4566-4575).

126. Vidal, R., Bruna, J., Giryes, R. and Soatto, S., 2017. Mathematics of deep learning. *arXiv preprint arXiv:1712.04741*.

127. Wang, Z., She, Q. and Ward, T.E., 2019. Generative Adversarial Networks: A Survey and Taxonomy. *arXiv preprint arXiv:1906.01529*.

128. Weiss, K., Khoshgoftaar, T.M. and Wang, D., 2016. A survey of transfer learning. *Journal of Big data*, *3*(1), p.9.

129. Xie, Z., 2017. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534*.

130. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. and Le, Q.V., 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237*.

131. Yu, L., Zhang, W., Wang, J. and Yu, Y., 2017, February. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

132. Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D.J. and Mannor, S., 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems* (pp. 3562-3573).

133. Zhang, X. and Ma, H., 2018. Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations. *arXiv preprint arXiv:1801.10459*.

9 Appendix B – Original Project Proposal

Towards Neural Paraphrase Generation | Project Proposal

1 Introduction

The application of neural networks to Natural Language Processing (NLP) has revolutionized the field and achieved immense success, particularly as of late with the success of transfer learning models for NLP. (Young et al., 2018) Natural Language Generation (NLG) models have historically been rule based systems (Young et al., 2018) although with the success of sequence to sequence models and later variations using attention most NLG benchmarks have moved to neural network-based approaches. (Goldberg, 2018) While translation and image captioning have been the focus of much of the research into generative models, paraphrase generation and identification poses an important problem which can be solved using similar approaches. (Xu, 2014) Another major issue in applying machine learning to NLG is finding good automatic evaluation metrics which capture the underlying objective that can then be optimized. (Novikova et al., 2017; Liu et al., 2017; Vedantam et al., 2015; Anderson et al., 2016)

The field of Reinforcement Learning (RL) has experienced a significant amount of growth as a result of recent success from applying deep neural networks to challenging problems. Reinforcement Learning studies a class of approaches to find actions which maximize the total reward an agent receives as it interacts with its environment (Sutton and Barto, 1998). While RL agents can be trained to perform well in complex environments such as Go (Silver et al., 2016; Silver et al., 2017), Atari (Mnih et al., 2013), and StarCraft II (DeepMind 2019) they have not been widely applied to NLP problems (with the exception of dialogue systems).

This research will be conducted in partnership with Phrasee, which is a short-form language generation company that focuses on applications to marketing.

1.1 Research Question

Given the success in applying neural networks to NLG and the advancements in reinforcement learning, two complimentary research questions were developed:

- ***What is the best sequence to sequence architecture for paraphrase generation?***

- ***What is the most effective reinforcement learning reward function for paraphrase generation?***

1.2 Aim

The aim of this project is to develop a paraphrase generation framework which first trains a general paraphrase generation model that is then fine-tuned using reinforcement learning for the desired performance objective.

1.3 Objectives

1. Objective	Testable Result
Thorough literature review of sequence to sequence models including paraphrase generation / identification Thorough literature review of RL approaches for NLP	Comprehensive "Context" section in final project report
Review of pretrained sentence encoder models	Evaluation of performance of models with various SoA sentence encoder models
Train GRU model on paraphrase sentence pairs	Trained paraphrase generation model using a specified encoder and decoder model
Fine-tune RL language model on specified objective (reward function)	Trained RL paraphrase generation model fine-tuned model on reward resulting in improved performance
Identify best existing metric for evaluating performance in paraphrase generation	Use of principled metric with theoretical justification to assess model performance
Contribute to "world's body of knowledge" (Dawson, 2009, p. 17)	Research demonstrating the best approach in applying reinforcement learning to fine-tune paraphrase generation models
Develop GUI / tool Phrasee can use to generate sentential paraphrases	Tool in which short form text is entered and optimized text is returned with the same semantic meaning
Develop list of future projects which build off this work	Comprehensive 'Future Works' section in final project report

1.4 Work Products

The project is intended to deliver the following products:

- A tool for Phrasee to create better performing short form text.
- A principled approach that practitioners wanting to use RL for NLP can follow.
- A comparison of supervised and RL approaches to paraphrase generation.

1.5 Project Beneficiaries

- **Phrasee:** generating and evaluating high performing subject lines and other short form text is core to Phrasee's business. A model which could improve existing approaches would significantly benefit Phrasee and generate substantial revenue.

- **Reinforcement learning research:** while RL has been very successful in many domains, it has been underapplied to language problems. This is partially due to the fact that language has an immense state space which is problematic for many out of the box RL algorithms. This project seeks to "contribute to the world's body of knowledge" (Dawson, 2009, p. 17) by developing a framework other researchers can follow to apply RL agents to NLP problems.

2 Critical Context

The combination of improvements in computational power and RL algorithms has led to RL agents achieving superhuman performance in complex environments including Go (Silver et al., 2016; Silver et al., 2017), Atari (Mnih et al., 2013), and StarCraft II (DeepMind, 2019). This has led to an increased interest in applying RL to the many NLP problems which can be formulated as MDPs (Young et al., 2018). There are several advantages to applying RL approaches to NLP problems including an ability to optimize for non-differentiable loss functions directly such as BLEU and ROUGE, facilitating actor-critic training (Bahdanau et al., 2016; Kumar et al., 2018), and most importantly incorporating the interactive dynamics of conversation which are crucial to dialogue systems.

Some successful applications of RL to NLP include dialogue systems (Li et al., 2017; Zhao et al., 2016), image captioning (Xu et al., 2015), text summarization (Paulus et al., 2017; Zhang et al., 2017), and text generation (Ranzato et al., 2015; Yu et al., 2017).

However, there are also significant challenges in applying RL algorithms to NLP tasks including formulating the problem correctly, ensuring the agent is able to learn in the massive action space, and the significant computation required. Efficient transfer learning would help many of these problems and in a sense is what is done when initializing the policy with a pretrained supervised policy as is done in AlphaGo (Silver et al., 2016) and MIXER (Ranzato et al., 2015).

The main novelty introduced by MIXER is an approach to handling the large action space through initializing a REINFORCE policy with a pretrained supervised model and gradually unfreezing the weights to leverage the RL agent's predictions. The other is solving the problem of exposure bias in which the model is only exposed to the training distribution rather than its own predictions as is done in a generative setting during test time. This is addressed through using the model's predictions at training time and optimizing for BLEU directly rather than cross-entropy.

Another significant paper which improves upon MIXER is Improved Image Captioning via Policy Gradient optimization of SPIDEr (Liu et al., 2017). Liu propose an approach wherein they use a supervised model trained with MLE to warm-start the reinforcement learning model then use an actor-critic architecture wherein the critic provides estimated value rewards. This creates an approach which can optimize for any reward function (including non-differentiable) and achieves state of the art performance on MS-COCO image captioning. They also propose SPIDEr which is a linear combination of SPICE and CIDER automatic metrics as an improvement over an individual metrics for optimization.

Applying neural networks to paraphrase identification and generation has been previously explored (Lan et al., 2018; Yin 2015 et al.; Xu 2014; Zhang et al., 2017) and applying reinforcement learning to paraphrase generation specifically has been explored by Li et al. (2018). Li et al. use a generator-discriminator framework wherein the generator is a sequence to sequence model which is trained using supervised learning and then RL and the discriminator is a deep matching model. Li et al. achieve state of the art performance using inverse reinforcement learning as a reward function. This approach is powerful as defining a reward function for paraphrase generation is challenging given widely used automatic metrics are not very good at approximating human judgment. (Novikova et al., 2017; Liu et al., 2017; Vedantam et al., 2015; Anderson et al., 2016)

3 Approaches: Methods & Tools for Design, Analysis, and Evaluation

As this project involves a reasonable level of computational resources, will result in a tangible work product, and needs to remain flexible to changes as research is released and better approaches are potentially discovered, an iterative software model is the most appropriate. IBM's Rational Unified Process (RUP) outlines four phases of development and engineering workflows with building blocks. The phases are expanded on in the Work Plan section as well as the visual project architecture.

- ***Inception Phase:*** this proposal can be used to achieve the first RUP phase which outlines the project feasibility and high-level requirements.
- ***Elaboration Phase:*** serves to further analyze and refine the requirements and will include a thorough literature review, increasing familiarization with existing Python implementations of sentence embedding models on GitHub including: InferSent, BERT, and others, and increasing familiarity with proposed datasets including: MS-COCO, Quora duplicate questions, Twitter shared URLs, and Wiki duplicate questions.
- ***Construction Phase:*** where the coding and implementation takes place. The code will be developed in Python with the use of PyTorch for deep learning as recommended by the teaching assistants. Testing and model evaluation will also take place at this stage. Details about the workflows and building blocks are contained in the visual project architecture below and also in the Work Plan.
- ***Transition Phase:*** where the final product is released and delivered to customers and maintenance plan. The current plan is not to deploy the model explicitly but rather to extract the key insights from the previous phases into the report for later implementation.

Visual Project Architecture

Proposed Development Modules				
Create Dataset	**Embed Source Sentences**	**Train Supervised Decoder**	**Fine-tune Reinforcement Learning Model**	**Evaluate Performance**
▪ Create dataset combining several sources and weighting each as appropriate	▪ Embed source sentences using a variety of approaches to determine best performance	▪ Train decoder using maximum likelihood estimation and teacher forcing with the labels from the dataset	▪ Initialize reinforcement learning policy decoder using the supervised model weights and fine-tune each model for each metric	▪ Evaluate performance of converged RL model optimized for specific metric across metrics and compare to other models
• Input: .txt files • Load data, perform preprocessing including ensuring max lengths • Output: formatted source-target paraphrase pairs and populated vocabulary	• Input: formatted paraphrase pairs • Embed source sentence using selected embedder • Output: sentence embeddings in vector or matrix form	• Input: sentence embedding • Train decoder model on dataset using MLE • Output: trained model weights on dataset	• Input: supervised model • Update weights through optimizing the policy for selected reward function • Output: fine-tuned model which achieves improved performance	• Input: RL model • Evaluate model performance on other metrics • Output: analysis of how specific model performs across other metrics

The general approach to achieving the results consists of first selecting a dataset. After evaluating different datasets previously used in paraphrasing tasks, I feel that the MS-COCO image captioning dataset is the best dataset to use until a working pipeline is completed. The Twitter data and Wiki question data has too much noise and the Quora duplicate question data may be added later although contains a subset of language which is very different from the image caption data thereby confusing the model. While the pipeline is being developed the dataset will be limited to 20k examples which will be expanded to ~200k samples for model benchmarking and hyperparameter tuning with the final model training on the full 1.2mm samples.

The Visual Project Architecture follows the ordering of the tasks to be completed in conjunction with the literature review. Multiple models have been identified as potential encoders with InferSent, BERT, and GloVe embeddings currently implemented. A vanilla encoder model has also been implemented. Additional models will be tested with the reinforcement learning model taking the encoder as a black box input.

The supervised decoder will also have different architectures tested although the current best performing architecture is a 256-node single layer GRU. A decoder with attention has also been implemented although further testing is required to determine the best performing model. Both decoder architectures will be first trained using supervised learning (maximum likelihood estimation). The best model is that which has the lowest error on the validation set.

The supervised model will then be transferred to a reinforcement learning agent with the same architecture. There are currently two RL agents implemented: a REINFORCE policy-based model, and an actor-critic policy/value-based model. For the REINFORCE model the agent is just taken from the supervised model to start and gradually deviates per MIXER. For the actor-critic agent, the actor begins with the supervised model then gradually deviates similar to MIXER while the critic is a trained model that estimates the value of each state in order to improve the credit assignment problem. SPIDEr uses monte carlo tree search which may also be implemented to improve performance. In order to improve learning, the currently contemplated reward scheme is the delta between the supervised learning model and the reinforcement learning model. (E.g. if the supervised model achieves 0.25 BLEU for a paraphrase generated using MLE and the RL agent achieves 0.30 BLEU then the reward will be 0.05)

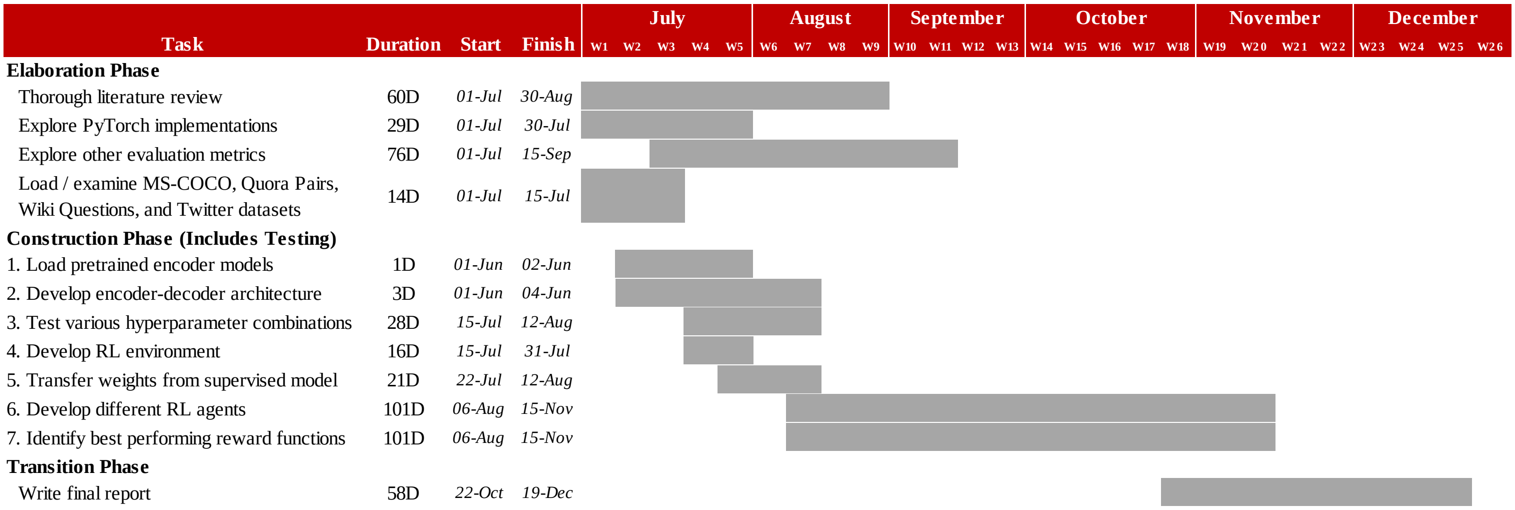
3.1 Evaluation of Results and Comparison to Alternative Approaches

Unfortunately, as paraphrase generation is a niche problem within natural language processing there are no established benchmarks or metrics for which to compare performance to.

Performance between the various supervised learning model configurations with the reinforcement learning models trained on different reward functions will be evaluated on existing automatic metrics such as BLEU, ROUGE, and METEOR, with image captioning data also evaluated using SPICE, CIDEr, and SPIDEr. One of the focuses of the project will be the extent to which these automatic metrics capture the task of measuring how effectively a paraphrase has been generated or if inverse reinforcement learning is a better approach to follow similar to Li et al., 2018.

The model performance can be benchmarked against Li et al., 2018 by retraining using the Quora dataset or through creative engineering and comparing the scores of generated paraphrases to identified paraphrases using Lan et al.'s (2017) paraphrase discriminator.

4 Work Plan



Task	Duration	Start	Finish
Elaboration Phase			
Thorough literature review	60D	*01-Jul*	*30-Aug*
Explore PyTorch implementations	29D	*01-Jul*	*30-Jul*
Explore other evaluation metrics	76D	*01-Jul*	*15-Sep*
Load / examine MS-COCO, Quora Pairs, Wiki Questions, and Twitter datasets	14D	*01-Jul*	*15-Jul*
Construction Phase (Includes Testing)			
1. Load pretrained encoder models	1D	*01-Jun*	*02-Jun*
2. Develop encoder-decoder architecture	3D	*01-Jun*	*04-Jun*
3. Test various hyperparameter combinations	28D	*15-Jul*	*12-Aug*
4. Develop RL environment	16D	*15-Jul*	*31-Jul*
5. Transfer weights from supervised model	21D	*22-Jul*	*12-Aug*
6. Develop different RL agents	101D	*06-Aug*	*15-Nov*
7. Identify best performing reward functions	101D	*06-Aug*	*15-Nov*
Transition Phase			
Write final report	58D	*22-Oct*	*19-Dec*

The Gantt chart above outlines an illustrative timeline for the project through the RUP phases. Steps 1-4 are computationally intensive although lower risk given the number of online tutorials and existing research. As previously stated, the main work that will take place is in steps 5-7 as they possess the most novelty and computational requirements. Writing the final report is likely to be done while the models are training in practice although to ensure sufficient time, from the end of October until the deadline has been dedicated to this task.

5 Risks

Risk Description	Prob.	Imp.	Value	Response to Risk
Inability to produce meaningful results with reinforcement learning models	0.35	0.60	0.21	This is the main project risk as deep RL models tend to be very unstable. However, REINFORCE has been used in multiple NLP applications and in the event it does not produce meaningful results an analysis of its limitations will be conducted.
Inability to produce meaningful results with supervised learning models	0.25	0.60	0.15	Try different model architectures and leverage pretrained models (e.g. AWD-LSTM). This is unlikely given success in research and success achieved thus far.
Research is released that is similar to this project	0.35	0.20	0.07	Will read the research which is released and incorporate any useful tricks that are introduced
Unable to find evaluation metric which captures objective	0.10	0.20	0.02	Create new metric or use combination of existing metrics
Models take too long to train / insufficient resources	0.25	0.20	0.05	Change source corpus to sample of dataset, leverage pretrained models for source models, downsample Simpsons character dialogue
Code or report is lost due to hard drive failure	0.05	0.50	0.03	This project will leverage GitHub and Google Drive to ensure minimal loss from potential hard drive failure
PyTorch is not sufficiently expressive or there are more RL resources in TensorFlow	0.05	0.10	0.01	Unlikely given established PyTorch RL community although will change to TensorFlow
Spending too much time on elaboration phase	0.15	0.10	0.02	Ensure strong time management and prioritization

6 Potential Extensions

There are several potential extensions to the project which will be considered time-permitting and will be otherwise be included as future work in the project.

- Training a RL algorithm from scratch rather than warm starting
- Defining an error function as difference between semantic meaning of predicted and ground truth using word embeddings (or sentence embeddings for sentence completion)
- Apply generator / discriminator framework to paraphrase generation task
- Applying the trained paraphrase generation model to different downstream NLP tasks as either a sentence embedder on the encoding side or pretrained discriminator in paraphrase identification

7 Ethical, Professional & Legal Issues

This project does not have any inherent ethical issues and does not require ethical approval per City, University of London's Research Ethics Review Form. The ethical, professional, and legal issues are more a consequence of improved paraphrase generation models. These can potentially be used in bot networks where an actor wants a message disseminated but to avoid detection can generate paraphrases of the content to create a false sense of consensus online.

It should be noted that OpenAI decided not to release their full trained language model as they felt there was a great risk it could be used for nefarious purposes such as generating misleading news articles, impersonating others online, and automating the production of spam email (OpenAI, 2019). While there is much contention around this decision, I feel that they are justified in restricting access to the model and would be prepared to do the same if the results end up being surprisingly strong.

A completed Research Ethics Review Form is attached in the appendix.

APPENDIX A: Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

http://www.city.ac.uk/department-computer-science/research-ethics

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to questions 1 – 18 and "yes" to question 19 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be provisional: the student may need to seek additional approval from the supervisor as the project progresses and details are established.

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online -		*Delete as appropriate*

https://ethics.city.ac.uk/		
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/*	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**

3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**NO**

REFERENCES

1. Anderson, P., Fernando, B., Johnson, M. and Gould, S., 2016, October. Spice: Semantic propositional image caption evaluation. In *European Conference on Computer Vision* (pp. 382-398). Springer, Cham.
2. Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A. and Bengio, Y., 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.
3. Dawson, C., 2009. *Introduction to Research Methods 5th Edition: A Practical Guide for Anyone Undertaking a Research Project*. Robinson.
4. DeepMind (2019) “AlphaStar: Mastering the Real-Time Strategy Game StarCraft II” Available at: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ (Accessed: 26 April 2019).
5. Goldberg, Y. 2018, “Neural Language Generation” [Presentation], International Conference on Natural Language Generation
6. Kumar, V., Ramakrishnan, G. and Li, Y.F., 2018. A framework for automatic question generation from text using deep reinforcement learning. *arXiv preprint arXiv:1808.04961*.
7. Lan, W. and Xu, W., 2018. Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering. *arXiv preprint arXiv:1806.04330*.
8. Lan, W., Qiu, S., He, H. and Xu, W., 2017. A continuously growing dataset of sentential paraphrases. *arXiv preprint arXiv:1708.00391*.
9. Li, X., Chen, Y.N., Li, L., Gao, J. and Celikyilmaz, A., 2017. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*.
10. Li, Y., 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

11. Li, Z., Jiang, X., Shang, L. and Li, H., 2017. Paraphrase generation with deep reinforcement learning. *arXiv preprint arXiv:1711.00279*.
12. Liu, S., Zhu, Z., Ye, N., Guadarrama, S. and Murphy, K., 2017. Improved image captioning via policy gradient optimization of spider. In *Proceedings of the IEEE international conference on computer vision* (pp. 873-881).
13. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
14. Novikova, J., Dušek, O., Curry, A.C. and Rieser, V., 2017. Why we need new evaluation metrics for NLG. *arXiv preprint arXiv:1707.06875*.
15. Open AI, 2019. Better Language Models and Their Implications. *Available at:* https://openai.com/blog/better-language-models/ (Accessed: 26 April 2019).
16. Paulus, R., Xiong, C. and Socher, R., 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.
17. Ranzato, M.A., Chopra, S., Auli, M. and Zaremba, W., 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.
18. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), p.484.
19. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of go without human knowledge. Nature, 550(7676), p.354.
20. Sutton, R.S. and Barto, A.G., 1998. Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.
21. Vedantam, R., Lawrence Zitnick, C. and Parikh, D., 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4566-4575).
22. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R. and Bengio, Y., 2015, June. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048-2057).
23. Xu, W., 2014, Data-driven approaches for paraphrasing across language variations. *PHD Dissertation*
24. Yin, W. and Schütze, H., 2015. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*(pp. 901-911).
25. Young, T., Hazarika, D., Poria, S. & Cambria, E. 2018, "Recent Trends in Deep Learning Based Natural Language Processing [Review Article]", IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 55-75.
26. Yu, L., Zhang, W., Wang, J. and Yu, Y., 2017, February. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*
27. Zhang, X. and Lapata, M., 2017. Sentence simplification with deep reinforcement learning. *arXiv preprint arXiv:1703.10931*.
28. Zhao, T. and Eskenazi, M., 2016. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. *arXiv preprint arXiv:1606.02560*.

8 Appendix C – Sample of Generated Text by Each Model

Input Sentence	Target Sentence	Supervised Model	BLEU-1	BLEU-2	ROUGE	CIDEr
a man eating food while holding a badminton racquet	a man with a badminton racket eating food .	a man is eating a in in the .	a man is holding a in a the .	a man holding a frisbee in a kitchen .	a man is holding a in a .	a man that is sitting on a table .
a dog at the entrance of a cluttered living room .	a dog walking into the living room alone	a dog is sitting in a chair looking room	a dog is sitting on a in a .	a dog sitting on a bed in a living room .	a dog is sitting on a in a .	a dog that is sitting on a table .
four people on a beach with surf boards	a family on the beach points into the water .	a few people walking on the beach with their surfboards	a group of people on a beach in the .	a group of people on a beach .	a group of people on a beach .	a couple of people on a beach .
two people walk side by side under an umbrella .	two people walking on a street holding an umbrella	two people walking down the street with an umbrella	two people walking on a in the street .	two people standing in a street with a .	two people walking on a street in a .	a couple of people walking on a street .
a surfer is at the crest of a long wave .	a person riding a surf board on a wave	a person riding a surfboard in the ocean .	a man is riding a on in the .	a man riding a wave on a surfboard .	a man on a surfboard in the water .	a person riding a wave on a wave .
an egyptian airlines plane landing at an airport .	a commercial plane on the strip to take off .	a large airplane is on runway on the runway .	a plane is on a in the the .	a plane is on a runway on a runway .	a airplane is on a runway .	a airplane on a runway on a runway .
a person is holding a huge scissor with his hands	a man is standing holding a large pair of scissors .	a man is holding his hands in his hand	a man is holding a in the on .	a man holding a cell phone in a kitchen .	a man is holding a in a .	a man holding a a on a .
a pile of different style and colored bags of luggage .	crowded baggage pick up point in an airport .	a pile of luggage on a wooden surface .	a luggage of luggage on a in the .	a luggage of luggage on a wooden table .	a luggage of luggage on a wooden table .	a couple of of luggage on a table .
three giraffes in an enclosure eat food from a trough .	some giraffes standing next to each other in their pen	three giraffes are standing together eating from a tree .	a group of giraffes are standing in the .	a group of giraffes standing in a field .	a group of giraffes standing in a .	a couple of giraffes that are on a field .
a bathroom with a sink and television in it	a person takes a picture in a hotel bathroom .	a bathroom with a sink and a the	a bathroom with a sink and a in the .	a bathroom with a sink in a bathroom .	a bathroom with a sink and a .	a bathroom with a sink a and a .
a couple of men eating hot dogs on wrappers .	two mean facing each other eating hot dogs .	two people are eating a hot on a table .	two people of two dogs on a in .	two people are sitting on a table .	two people of two dogs on a .	a couple of people that is on a table .
the bathroom has a sink and see through shower .	modern bathroom with exotic tiling and counter top .	a bathroom with a sink shower and toilet	a bathroom with a sink and and a .	a bathroom with a sink in a bathroom .	a bathroom with a sink and a .	a bathroom with a shower toilet and a .
a small group of elephants stand together in the grass .	two adult elephants and a baby elephant in a field .	a herd of elephants are standing in field	a group of elephants in a field in the .	a group of elephants standing in a field .	a group of elephants in a field .	a couple of elephants in a field .
several people with numbers going downhill on skis .	a group of cross country skiers skiing on a trail	a group of people are skiing down some skis	a group of people on a in the .	a group of people are skiing on a snowy hill .	a group of people on a in the .	a person that is sitting on a train .
a zebra standing on top of a dry grass field .	the zebra does not look to be fully grown .	a zebra standing in a grassy field with trees	a zebra is standing on a in the .	a zebra standing in a field in a field .	a zebra standing in a field in a .	a zebra walking in a field on a field .
children playing with a frisbee by a mountain lake	a lot of kids are at the park	two boys playing with a frisbee in the grass .	a group of people playing in a the .	a group of people are playing a field .	a group of young children playing a frisbee .	a group of people playing on a field .
nine boys sitting outside on the front steps	a group of boys sit on steps with sport equipment .	a group of people sitting on on on the . .	a group of people on a in the .	a group of people sitting on a bench .	a group of people on a in the .	a couple of people on a bench .
a gray elephant with small tusks walking around	a large gray elephant walking next to a building .	a large elephant is standing in front of a	a large elephant is standing in a the .	a man standing on a elephant in a field .	a large elephant is in a in the .	a couple of people that is on a table .

Input Sentence	Target Sentence	Supervised Model	PARA	PARA-F	PARASIM	PARASIM-F
a man eating food while holding a badminton racquet	a man with a badminton racket eating food .	a man is eating a in in the .	a man eating a plate with a doughnut	a man in the middle of the pizza .	a man holding a plate with a doughnut	a man in the middle of the kitchen .
a dog at the entrance of a cluttered living room .	a dog walking into the living room alone	a dog is sitting in a chair looking room	a dog sitting on a bed with a bed	a dog in the middle of the bed .	a dog sitting on a couch with a dog	a dog sitting in the middle of the bed .
four people on a beach with surf boards	a family on the beach points into the water .	a few people walking on the beach with their surfboards	several people on beach beach with people on beach	several people in the middle of the beach .	people on a beach with people on the beach	two people on the beach on the beach .
two people walk side by side under an umbrella .	two people walking on a street holding an umbrella	two people walking down the street with an umbrella	a woman walking on a street with umbrella	two people in the middle of the street .	two people walking on a street with umbrella	two people in the middle of the street .
a surfer is at the crest of a long wave .	a person riding a surf board on a wave	a person riding a surfboard in the ocean .	a man riding a wave on surfboard on surfboard	a man in the middle of the ocean .	a man riding a wave on a surfboard on ocean	a man on the surfboard on the waves .
an egyptian airlines plane landing at an airport .	a commercial plane on the strip to take off .	a large airplane is on runway on the runway .	a plane on a runway on a runway	a plane in the middle of the airport .	a plane on a runway on a runway runway	a plane on the side of the runway .
a person is holding a huge scissor with his hands	a man is standing holding a large pair of scissors .	a man is holding his hands in his hand	a man holding a cell phone with a cell	a man in the middle of the computer .	a man holding a cell phone with a cell	a man in the middle of the phone .
a pile of different style and colored bags of luggage .	crowded baggage pick up point in an airport .	a pile of luggage on a wooden surface .	a luggage bag on a plate with a	a luggage in the middle of the airport .	a luggage of luggage on a luggage on a table	a bunch of luggage on the side of the table .
three giraffes in an enclosure eat food from a trough .	some giraffes standing next to each other in their pen	three giraffes are standing together eating from a tree .	three giraffes standing in a field with trees	two giraffes in the middle of the field .	three giraffes are in a field with trees	three giraffes in the middle of the field .
a bathroom with a sink and television in it	a person takes a picture in a hotel bathroom .	a bathroom with a sink and a the	a bathroom with a sink a with a	a bathroom in the middle of the bathroom .	a bathroom with a sink with a toilet	a bathroom in the middle of the bathroom .
a couple of men eating hot dogs on wrappers .	two mean facing each other eating hot dogs .	two people are eating a hot on a table .	two people eating a plate with food on a table	two men in the middle of the table .	two people of people eating two hot dogs on a table	two men in the middle of the table .
the bathroom has a sink and see through shower .	modern bathroom with exotic tiling and counter top .	a bathroom with a sink shower and toilet	a bathroom with a sink toilet and bathroom	a bathroom in the middle of the bathroom .	a bathroom with a sink with toilet with shower	a bathroom in the middle of the bathroom .
a small group of elephants stand together in the grass .	two adult elephants and a baby elephant in a field .	a herd of elephants are standing in field	a herd of elephants standing elephants on grass	two elephants in the middle of the field .	two elephants in a field with elephants in a field	two elephants in the middle of the field .
several people with numbers going downhill on skis .	a group of cross country skiers skiing on a trail	a group of people are skiing down some skis	a group of people skiing on snow on	several skiers in the middle of the snow .	people are skiing on a snowy hill on a snowy	a skier in the middle of the snow
a zebra standing on top of a dry grass field .	the zebra does not look to be fully grown .	a zebra standing in a grassy field with trees	a zebra standing on a field with grass	a zebra in the middle of the field .	a zebra zebra standing in a field with trees	a zebra standing in the middle of the field .
children playing with a frisbee by a mountain lake	a lot of kids are at the park	two boys playing with a frisbee in the grass .	two kids playing frisbee on a field with grass	two boys in the middle of the field .	two people playing frisbee on a field with a frisbee	two boys in the middle of the field .
nine boys sitting outside on the front steps	a group of boys sit on steps with sport equipment .	a group of people sitting on on on the . .	a group of people on a on on skateboard	a skateboarder in the middle of the air .	a group of people sitting on a bench on a	a baseball player on the side of the bench .
a gray elephant with small tusks walking around	a large gray elephant walking next to a building .	a large elephant is standing in front of a	a elephant elephant with a elephant on a	a elephant in the middle of the field .	a elephant elephant in a elephant in a field	a elephant in the middle of the field .

Input Sentence	Target Sentence	Supervised Model	ESIM	Adversarial	Length Penalty
a man eating food while holding a badminton racquet	a man with a badminton racket eating food .	a man is eating a in in the .	a man is eating a in in the	a man is eating something in the kitchen .	a man is eating something
a dog at the entrance of a cluttered living room .	a dog walking into the living room alone	a dog is sitting in a chair looking room	a dog is sitting in a living room	a dog is playing with a chair in the room .	a dog is sitting in a chair
four people on a beach with surf boards	a family on the beach points into the water .	a few people walking on the beach with their surfboards	people of people beach with people in the beach	several people are walking on the beach .	four people with surfboards and people
two people walk side by side under an umbrella .	two people walking on a street holding an umbrella	two people walking down the street with an umbrella	two people walking in an umbrella in the park	two people walking down the street with an umbrella .	two people walking down the street
a surfer is at the crest of a long wave .	a person riding a surf board on a wave	a person riding a surfboard in the ocean .	a man is riding a surfboard in the water	a man riding a wave on a surfboard .	a person riding a wave on a
an egyptian airlines plane landing at an airport .	a commercial plane on the strip to take off .	a large airplane is on runway on the runway .	a airplane parked at a runway in an airport	an airplane and an airplane at an airport .	an airplane and an airplane
a person is holding a huge scissor with his hands	a man is standing holding a large pair of scissors .	a man is holding his hands in his hand	a man is holding a cell in the hands	the man is holding his hands in the air .	a man holding up a cell
a pile of different style and colored bags of luggage .	crowded baggage pick up point in an airport .	a pile of luggage on a wooden surface .	a luggage of luggage sitting on a table	a pile of luggage bags are sitting on a table .	a luggage full of luggage
three giraffes in an enclosure eat food from a trough .	some giraffes standing next to each other in their pen	three giraffes are standing together eating from a tree .	three giraffes of a giraffes standing in an enclosure	three giraffes are standing together by a fence .	three giraffes are eating hay
a bathroom with a sink and television in it	a person takes a picture in a hotel bathroom .	a bathroom with a sink and a the	a bathroom with a sink in the bathroom	a bathroom with the sink and a toilet .	a bathroom with a sink and
a couple of men eating hot dogs on wrappers .	two mean facing each other eating hot dogs .	two people are eating a hot on a table .	two people of people eating food in the park	two people are eating hot dogs on a table .	two people with glasses eating
the bathroom has a sink and see through shower .	modern bathroom with exotic tiling and counter top .	a bathroom with a sink shower and toilet	a bathroom with a shower shower in a bathroom	a bathroom with a sink and shower .	a bathroom with sink shower and toilet
a small group of elephants stand together in the grass .	two adult elephants and a baby elephant in a field .	a herd of elephants are standing in field	three elephants of elephants in a field	three elephants are standing in the grass field .	three elephants are standing in field
several people with numbers going downhill on skis .	a group of cross country skiers skiing on a trail	a group of people are skiing down some skis	a group of people skiing in an old snowy ski	several people are skiing down a snowy hill .	several people are skiing down
a zebra standing on top of a dry grass field .	the zebra does not look to be fully grown .	a zebra standing in a grassy field with trees	a zebra standing in a field in the dirt	a zebra standing in a grassy field .	a zebra standing in a grassy field
children playing with a frisbee by a mountain lake	a lot of kids are at the park	two boys playing with a frisbee in the grass .	two people playing frisbee in the field with the water	two young boys playing with a frisbee in the grass .	two boys playing with a frisbee
nine boys sitting outside on the front steps	a group of boys sit on steps with sport equipment .	a group of people sitting on on on the . .	a group of people sitting on a skateboard	three boys are sitting on a bench .	three boys sitting on the ground
a gray elephant with small tusks walking around	a large gray elephant walking next to a building .	a large elephant is standing in front of a	a elephant is in a elephant in the	a large elephant is standing in front of a .	a large elephant is standing in

9 Appendix D – Code Submission

The full code base is available in either of the following links:

- **https://cityuni-my.sharepoint.com/:f:/g/personal/andrew_gibbs-bravo_city_ac_uk/EsvigLjhwGhJqjCbrRO65ZoBBA11Ac1iIakvjzIFdcV4_g?e=6ctkw4**
- Backup link: **https://drive.google.com/open?id=1HiTMognmqe7nnU0cBeQr-Lsc-1r5uD1Z**

However, for ease of evaluation, we have included scripts for key modules (those that achieved the highest importance score) in section 3.2.

9.1 Data

```
"""Imports raw data from various sources, preprocesses, creates train/test sets and vocab index
Also contains functions for saving and loading"""

import pandas as pd
import numpy as np
import os
import pickle
import torch

import json
import itertools
import re
from collections import defaultdict

import config

DEVICE = config.DEVICE
MAX_LENGTH = config.MAX_LENGTH

def create_coco_pairs(input_path):
    """Loads MS-COCO data, gets caption data and converts to caption pairs"""
    # Load data from json
    with open(input_path) as json_data:
        data = json.load(json_data)

    # Instantiate dictionary
    captions_dict = defaultdict(list)

    # Fill dictionary with captions
    for item in data['annotations']:
        captions_dict[item['image_id']].append(item['caption'])

    # Pair captions and convert to list instead of tuple
    coco_pairs = []

    for _, value in captions_dict.items():
        coco_pairs.extend(itertools.combinations(value,2))

    coco_pairs = np.array([list(pair) for pair in coco_pairs])

    return coco_pairs

def create_quora_pairs(input_path):
    """Loads Quora data and keeps only questions labelled as duplicates as paraphrase pairs"""
    # Load Quora duplicates dataset
    df = pd.read_csv(input_path)

    # Keep only duplicate questions
    df = df[df['is_duplicate'] == 1]
```

```
    df.drop(['id','qid1','qid2','is_duplicate'], axis=1, inplace=True)

    quora_pairs = np.array([np.array([df['question1'].iloc[i],
                                   df['question2'].iloc[i]]) for i in range(len(df))])
    return quora_pairs

def create_pred_twitter_pairs(input_path, sim_threshold=0.75):
    """Loads automated twitter data and keeps only pairs over specified model confidence threshold
    as paraphrase pairs"""
    df = pd.read_csv(input_path, sep="\t",
                 header=None, usecols = [0,1,2])
    df.columns = ['sim_score', 'sent1', 'sent2']

    # Keep only captions with similarity greater than the threshold
    df = df.loc[df['sim_score']>sim_threshold]

    sentence_pairs = [[a,b] for a,b in df[['sent1','sent2']].values]
    return sentence_pairs

def create_human_twitter_pairs(input_path, sim_threshold=0.5):
    """Loads human labelled twitter data and keeps only pairs over inter-rater agreement threshold
    as paraphrase pairs"""
    df = pd.read_csv(input_path, sep="\t", header=None, usecols = [0,1,2])
    df.columns = ['sent1', 'sent2', 'sim_score']

    # Convert rater agreement into score
    df['sim_score'] = [int(i[1])/int(i[3]) for i in df['sim_score']]

    # Keep only captions with similarity greater than the threshold
    df = df.loc[df['sim_score']>sim_threshold]

    sentence_pairs = [[a,b] for a,b in df[['sent1','sent2']].values]

    return sentence_pairs

class VocabIndex:
    """Class which converts sentences to a vocabulary and gives each word an index as well as a
count"""
    def __init__(self):
        self.word2index = {'SOS':config.SOS_token, 'UNK':config.UNK_token, 'EOS':config.EOS_token}
        self.word2count = {'SOS':1, 'UNK':1, 'EOS':1}
        self.index2word = {0: "SOS", 1: "EOS", 2: "UNK"}
        self.n_words = 3  # Count SOS, EOS, and Unk

    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1

def preprocess(input_text, remove_punct=True, lower_case=False):
    """Preprocesses raw text based on required preprocessing steps"""
    # Convert to String
    input_text = str(input_text)

    # Lower first character
    input_text = input_text[0].lower() + input_text[1:]

    if remove_punct:
        # Add space in front of key punctuation and remove other punctuation
        input_text = re.sub(r"([.!?])", r" \1", input_text)
        input_text = re.sub(r"[^a-zA-Z.!?-]+", r" ", input_text)

    if lower_case:
        # Convert to lower case
        input_text = input_text.lower()

    return input_text
```

```python
def filterPairs(pairs, max_length=18):
    """Removes pairs where either sentence has more tokens than the maxmimum"""
    return np.array([pair for pair in pairs if
            len(pair[0].split(' ')) < max_length and \
            len(pair[1].split(' ')) < max_length])

def sample_list(input_list, n_samples=5000, sample_by_prop=False, sample_prop=0.10):
    """Returns a random subset of an input list based on either number of samples or proportion"""
    if sample_by_prop:
        selected_indices = np.random.choice(len(input_list),
                                            int(sample_prop * len(input_list)), replace=False)
    else:
        selected_indices = np.random.choice(len(input_list), n_samples, replace=False)
    sampled_list = input_list[selected_indices]
    return sampled_list

def caption_processing_pipeline(input_pairs, n_samples, max_length=18,
                                remove_punct=True, lower_case=False):
    """Applies filtering, preprocessing, and sampling to raw dataset"""
    filtered_pairs = filterPairs(input_pairs, max_length)
    for idx, pair in enumerate(filtered_pairs):
        filtered_pairs[idx][0] = preprocess(pair[0], remove_punct, lower_case)
        filtered_pairs[idx][1] = preprocess(pair[1], remove_punct, lower_case)

    refiltered_pairs = filterPairs(filtered_pairs, max_length)
    sampled_pairs = sample_list(refiltered_pairs, n_samples)
    return sampled_pairs

def convert_unk_terms(vocab_index, min_count=1):
    """Converts words below count level's index to UNK"""
    unk_words = [a for a,b in vocab_index.word2count.items() if b <= min_count]

    # Convert word to UNK index
    for word in unk_words:
        vocab_index.word2index[word] = config.UNK_token

    return unk_words, len(unk_words)

def get_pairs(dataset_size=20000, coco_prop=0.50, quora_prop=0.25, twitter_prop=0.25,
                      remove_unk=True, max_length=18):
    """Loads data based on dataset proportions and applies processing pipeline,
        then fills vocab_index and optionally removes unknown words"""
    # Load dataframes
    print('Loading data...')
    coco_train_pairs = create_coco_pairs(config.coco_train_path)
    coco_val_pairs = create_coco_pairs(config.coco_val_path)
    coco_pairs = np.vstack([coco_train_pairs, coco_val_pairs])

    quora_pairs = create_quora_pairs(config.quora_path)

    twitter_pairs = create_pred_twitter_pairs(config.twitter_path)

    # Filter out sentences greater than specified length and downsample
    # Can also be used to remove data from sample through setting sample_prop to zero
    sampled_coco_pairs = caption_processing_pipeline(coco_pairs, int(dataset_size*coco_prop),
                                                      max_length, remove_punct=True, lower_case=True)
    sampled_quora_pairs = caption_processing_pipeline(quora_pairs, int(dataset_size*quora_prop),
max_length)
    sampled_twitter_pairs = caption_processing_pipeline(twitter_pairs, int(dataset_size*twitter_prop),
                                                        max_length, remove_punct=True, lower_case=True)

    # Merge dataframes
    caption_pairs = np.vstack([sampled_coco_pairs, sampled_quora_pairs, sampled_twitter_pairs])

    # Initialize VocabIndex and populate
    caption_vocab_index = VocabIndex()

    for idx, pair in enumerate(caption_pairs):
        caption_vocab_index.addSentence(pair[0])
        caption_vocab_index.addSentence(pair[1])

    print('Dataframe successfully created:')
    print('Total Samples: {}'.format(len(caption_pairs)))
    print('    - COCO Image Captioning Samples: {} ({:.1%})'.format(len(sampled_coco_pairs),
          len(sampled_coco_pairs) / len(caption_pairs)))
```

```
    print('    - Quora Duplicate Question Samples: {} ({:.1%})'.format(len(sampled_quora_pairs),
          len(sampled_quora_pairs) / len(caption_pairs)))
    print('    - Twitter Share URL Samples: {} ({:.1%})'.format(len(sampled_twitter_pairs),
          len(sampled_twitter_pairs) / len(caption_pairs)))

    # Convert terms only occurring n times to unk in dict (DOES NOT IMPACT ACTUAL TEXT)
    if remove_unk:
        _, n_unk_words = convert_unk_terms(caption_vocab_index, min_count=1)

        print('Total vocabulary size: {}'.format(caption_vocab_index.n_words))
        print('{} UNK words'.format(n_unk_words))

    return caption_pairs, caption_vocab_index

def train_test_split(input_array, splits=(0.65,0.25,0.10)):
    """Creates a random train, val, test split for a given data input array"""
    np.random.seed(config.SEED)
    np.random.shuffle(input_array)
    n_dataset = len(input_array)
    train_split, val_split, test_split = splits

    train_idx = int(train_split * n_dataset)
    val_idx = int(train_split * n_dataset)+int(val_split * n_dataset)

    train_set = input_array[:train_idx]
    val_set = input_array[train_idx:val_idx]
    test_set = input_array[val_idx:]

    assert len(train_set) + len(val_set) + len(test_set) == len(input_array), "Some data has been lost"

    return train_set, val_set, test_set

def create_data(load_data=True, pairs_input_path='Data/pairs_data.npy',
                index_input_path='Data/vocab_index.pickle', dataset_size=20000):
    """Creates a train / test split and vocab_index by loading or
    creating the dataset based on the file path or specified dataset proportions / requirements"""
    if load_data:
        print('Loading saved dataset....')
        pairs = load_np_data(pairs_input_path)
        vocab_index = load_vocab_index(index_input_path)
        print('Dataset loaded.')
        print('    Total number of sentence pairs: {}'.format(len(pairs)))
        print('    Total vocabulary size: {}'.format(vocab_index.n_words))

    else:
        # Create dataset from corpora
        pairs, vocab_index = \
        get_pairs(dataset_size=dataset_size, coco_prop=1,
                                            quora_prop=0, twitter_prop=0,
                                            remove_unk=False, max_length=MAX_LENGTH)
        # Shuffle dataset and ensure
        np.random.seed(config.SEED)
        np.random.shuffle(pairs)
        assert max([max(len(a.split()),len(b.split())) for a,b in pairs]) < MAX_LENGTH, "Pairs exceed
MAX_LENGTH"

    train_pairs, val_pairs, test_pairs = train_test_split(pairs, splits=(0.65,0.25,0.10))
    return train_pairs, val_pairs, test_pairs, vocab_index, pairs

def instantiate_vocab_idx(input_path):
    """Instantiates a vocab_index and fills it with the caption pairs"""
    caption_pairs = load_np_data(input_path)
    caption_vocab_index = VocabIndex()

    for idx, pair in enumerate(caption_pairs):
        caption_vocab_index.addSentence(pair[0])
        caption_vocab_index.addSentence(pair[1])
    return caption_vocab_index

# %% Saving and Loading Data

def save_np_data(input_data, file_name):
    """Only designed for saving Numpy arrays therefore name must include .npy extension"""
    if os.path.isfile(file_name):
        print('Error: File already exists - please change name or remove conflicting file')
    else:
```

```
        assert '.npy' in file_name, 'Please ensure .npy extension is included in file_name'
        np.save(file_name, input_data)

def load_np_data(file_name):
    """Only designed for loading Numpy arrays"""
    assert '.npy' in file_name, 'Please ensure file is .npy filetype'
    return np.load(file_name)

def save_np_to_text(input_data, file_name):
    """Designed for saving txt files therefore name must include .txt extension"""
    assert '.txt' in file_name, 'Please ensure .txt extension is included in file_name'
    with open(file_name, 'a') as file:
        np.savetxt(file, input_data, fmt='%1.2f')

def save_dict(input_dict, file_name):
    """Only designed for saving dicts to JSON arrays therefore name must include .json extension"""
    if os.path.isfile(file_name):
        print('Error: File already exists - please change name or remove conflicting file')
    else:
        assert '.json' in file_name, 'Please ensure .json extension is included in file_name'
        with open(file_name, 'w') as fp:
            json.dump(input_dict, fp)

def load_dict(file_name):
    """Used to load JSON dicts"""
    with open(file_name) as json_data:
        data = json.load(json_data)
    return data

def save_vocab_index(vocab_index, file_name):
    """Only designed for pickling the vocab idx therefore name must include .pickle extension"""
    if os.path.isfile(file_name):
        print('Error: File already exists - please change name or remove conflicting file')
    else:
        assert '.pickle' in file_name, 'Please ensure .pickle extension is included in file_name'
        pickle_out = open(file_name, "wb")
        pickle.dump(vocab_index, pickle_out)
        pickle_out.close()

def load_vocab_index(file_name):
    """Only designed for loading pickle files"""
    assert '.pickle' in file_name, 'Please ensure file is .pickle filetype'
    with open(file_name, 'rb') as handle:
        vocab_index = pickle.load(handle)
    return vocab_index


#%% Saving and Loading Models

def save_model(model, file_name):
    """Only designed for saving PyTorch model weights therefore must include .pt extension"""
    if os.path.isfile(file_name):
        print('Error: File already exists - please change name or remove conflicting file')
    else:
        assert '.pt' in file_name, 'Please ensure .pt extension is included in file_name'
        torch.save(model.state_dict(), file_name)

def load_model(model, file_name, device=DEVICE):
    """Only designed for loading PyTorch model weights therefore must ensure model has an
    identical structure to saved version"""
    if DEVICE.type == 'cuda':
        model.load_state_dict(torch.load(file_name))
        model.to(device)
    else:
        model.load_state_dict(torch.load(file_name, map_location=device))

def save_exp_args(exp_args, file_name):
    """Saves experiment input arguments from model runs"""
    args_dict = dict(vars(exp_args))
    save_dict(args_dict, file_name)

def save_model_args(input_model, file_name):
    """Save model arguments for each experiment"""
    try:
        model_dict = dict(vars(input_model)['_modules'])
        model_dict['model_name'] = input_model.name
```

```
        save_dict(str(model_dict), file_name)
    except:
        print("Unable to save model args")

def extract_model_number(input_text, start_symbol='_', end_symbol='.pt'):
    """Returns the model number for a given saved model"""
    m = re.search(start_symbol+'(.+?)'+end_symbol, input_text)
    if m:
        found = m.group(1)

    return float(found)

def get_top_n_models(input_path, model_type='decoder', n=1, descending=False):
    """Returns the the top n saved models by performance"""
    folder_files = os.listdir(input_path)
    loss_value = [extract_model_number(file) for file in folder_files if ('.pt' in file) \
                      and (model_type in file)]
    loss_value.sort(reverse=descending)
    n_values = loss_value[:n]
    return n_values

def load_RL_models(env_name, folder_name, actor_model, critic_model,
                   actor_file_name='best', critic_file_name='best'):
    """Loads saved RL models"""
    if actor_file_name == 'best':
        actor_file_name = 'actor_{:.3f}.pt'.format(
                get_top_n_models(
                        os.path.join(config.saved_RL_model_path, env_name, folder_name), 'actor', n=1,
descending=True)[0])

    load_model(actor_model, os.path.join(config.saved_RL_model_path, env_name,
                                              folder_name, actor_file_name))

    if critic_model is not None:
        if critic_file_name == 'best':
            critic_file_name = 'critic_{:.3f}.pt'.format(
                    get_top_n_models(
                            os.path.join(config.saved_RL_model_path, env_name, folder_name), 'critic',
n=1, descending=True)[0])

        load_model(critic_model, os.path.join(config.saved_RL_model_path, env_name,
                                                   folder_name, critic_file_name))

        return actor_model, critic_model

    else:
        return actor_model, None

class SaveSupervisedModelResults(object):
    """Object for storing supervised model results as the experiment is being run"""
    def __init__(self, folder_name):
        self.path = config.saved_supervised_model_path
        self.folder_name = folder_name
        self.folder_path = os.path.join(self.path, self.folder_name)

        self.track_loss = True
        self.train_loss = []
        self.val_loss = []
        self.val_loss_thresh = 3.70

    def check_folder_exists(self):
        if os.path.isdir(self.folder_path):
            raise Exception('This experiment folder already exists')

    def init_folder(self, exp_args, encoder_model=None, decoder_model=None):
        try:
            os.makedirs(self.folder_path)
        except FileExistsError:
            pass
        save_exp_args(exp_args, os.path.join(self.folder_path, 'exp_args.json'))
        save_model_args(decoder_model, os.path.join(self.folder_path, 'decoder_args.json'))

        if encoder_model is not None:
            save_model_args(encoder_model, os.path.join(self.folder_path, 'encoder_args.json'))

    def export_loss(self, train_file_name, val_file_name):
```

```
        save_np_to_text(self.train_loss, os.path.join(self.folder_path, train_file_name))
        save_np_to_text(self.val_loss, os.path.join(self.folder_path, val_file_name))
        self.reset()

    def save_top_models(self, input_model, file_name):
        save_model(input_model, os.path.join(self.folder_path, file_name))

    def reset(self):
        self.train_loss = []
        self.val_loss = []

class SaveRLModelResults(object):
    """Object for storing RL model results as the experiment is being run"""
    def __init__(self, env_name, folder_name):
        self.path = config.saved_RL_model_path
        self.folder_name = folder_name
        self.env_name = env_name
        self.folder_path = os.path.join(self.path, self.env_name, self.folder_name)

        self.env_rewards = []
        self.KL_penalty = []

    def check_folder_exists(self):
        if os.path.isdir(self.folder_path):
            raise Exception('This experiment folder already exists')

    def init_folder(self, exp_args, actor_model=None, critic_model=None):
        try:
            os.makedirs(self.folder_path)
        except FileExistsError:
            pass
        save_exp_args(exp_args, os.path.join(self.folder_path, 'exp_args.json'))
        save_model_args(actor_model, os.path.join(self.folder_path, 'actor_args.json'))

        if critic_model is not None:
            save_model_args(critic_model, os.path.join(self.folder_path, 'critic_args.json'))

    def export_rewards(self, file_name):
        if len(self.KL_penalty) > 0:
            combined_rewards = np.array([[reward, penalty] for (reward, penalty) in
                                      zip(self.env_rewards, self.KL_penalty)])
            save_np_to_text(combined_rewards, os.path.join(self.folder_path, file_name))

        else:
            save_np_to_text(self.env_rewards, os.path.join(self.folder_path, file_name))

        self.reset()

    def save_top_models(self, input_model, file_name):
        save_model(input_model, os.path.join(self.folder_path, file_name))

    def reset(self):
        self.env_rewards = []
        self.KL_penalty = []

def load_loss_data(folder_name):
    """Loads saved loss data for supervised models"""
    training_loss = pd.read_csv(os.path.join(config.saved_supervised_model_path, folder_name,
'training_loss.txt'),
                                sep=" ", header=None, names = ['train_loss'], dtype =
{'train_loss':np.float32})
    val_loss = pd.read_csv(os.path.join(config.saved_supervised_model_path, folder_name,
'val_loss.txt'),
                           sep=" ", header=None, names = ['val_loss'], dtype = {'val_loss':np.float32})

    n_iterations = int(len(training_loss)/len(val_loss))

    df = training_loss.copy()
    df['val_loss'] = np.repeat(val_loss['val_loss'].values, n_iterations)

    return df

def load_rewards_data(env_name, folder_name):
    """Loads saved rewards data for RL models"""
    rewards_df = pd.read_csv(os.path.join(config.saved_RL_model_path, env_name, folder_name,
                                          'model_performance.txt'), sep=" ", header=None)
```

```
    if len(rewards_df.columns) == 2:
        rewards_df.columns = ['env_rewards', 'KL_penalty']
        rewards_df['total_rewards'] = rewards_df['env_rewards'] + rewards_df['KL_penalty']

    elif len(rewards_df.columns) == 1:
        rewards_df.columns = ['env_rewards']

    return rewards_df

#%% ------------------------------ Load data ------------------------------
#Create data for use across all other modules. Loads the same saved data each time.

TRAIN_PAIRS, VAL_PAIRS, TEST_PAIRS, VOCAB_INDEX, _ = \
create_data(load_data=True, pairs_input_path=config.pairs_path,
                index_input_path=config.vocab_index_path, dataset_size=20000)

# Ensures that manually entered terms are in vocab_index
if 'SOS' not in VOCAB_INDEX.word2index:
    VOCAB_INDEX.word2index['SOS'] = config.SOS_token
    VOCAB_INDEX.word2index['UNK'] = config.UNK_token
    VOCAB_INDEX.word2index['EOS'] = config.EOS_token

    VOCAB_INDEX.word2count['SOS'] = 1
    VOCAB_INDEX.word2count['UNK'] = 1
    VOCAB_INDEX.word2count['EOS'] = 1


# Save data and index
#data.save_np_data(pairs, 'Data/pairs_data_100k.npy')
#data.save_vocab_index(vocab_index, 'Data/vocab_index_100k.pickle')

#%% -------------------------------------ARCHIVE----------------------------------------------

#def create_wiki_df():
#    """Creates dataframe for Wiki Answers dataset"""
#
#    # Duplicate questions after lemmatization
#    df = pd.read_csv('D:/Paraphrase_Datasets/WikiAnswers_Duplicates/word_alignments.txt',
#                     sep="\t", nrows=100000, header=None, usecols = [0,1])
#
#    # Finds the original question from the lemmatized version
#    question_df = pd.read_csv('D:/Paraphrase_Datasets/WikiAnswers_Duplicates/questions.txt',
#                     sep="\t",   header=None, usecols = [0,3])
#
#    search_text = 'how Dose the stellar'
#    original_question = question_df.iloc[question_df[3].loc[[search_text in str(q) for q in
question_df[3]]].index][0]
#    print(original_question.item())
#
#    return df
```

9.2 Encoder Models

```
"""Defines classes for each encoder: GloVe, BERT, InferSent,
Vanilla and GPT language model along with related code"""

import torch
import torch.nn as nn
import numpy as np
import math

import config

from pymagnitude import Magnitude
try:
    from pytorch_transformers import BertTokenizer, BertModel
    from pytorch_transformers import OpenAIGPTTokenizer, OpenAIGPTLMHeadModel
    from pytorch_transformers import GPT2Tokenizer, GPT2LMHeadModel

    from sentence_transformers import SentenceTransformer
except:
    print('Failed to import BERT, GPT, or GPT2')
```

```python
try:
    from InferSentModels import InferSent
except:
    print('Failed to import InferSent')

# Set device and pretrained models list
DEVICE = config.DEVICE
GPU_ENABLED = config.GPU_ENABLED

pretrained_models_list = ['GloveEncoder', 'InferSentEncoder', 'BERTEncoder', 'InitializedEncoder']

class EncoderRNN(nn.Module):
    """Encoder class which trains embeddings from scratch and specifies GRU architecture"""
    def __init__(self, input_size, embedding_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.name = 'RandomEncoderRNN'
        self.trainable_model = True
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, self.embedding_size)
        self.gru = nn.GRU(self.embedding_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

def create_vocab_tensors(input_vocab_index):
    """Creates a matrix of the glove embeddings for terms contained in the model for improve runtime
        Also used in ESIM"""
    print('Creating vocabulary tensors...')
    # Define GloVe model from Magnitude package
    model = Magnitude(config.glove_magnitude_path)

    np.random.seed(config.SEED)
    # Randomly initialize matrix
    vocab_tensors = np.random.normal(0, 1, (input_vocab_index.n_words, model.dim)).astype('float32')

    vocab_words = list(input_vocab_index.word2index.keys())
    unk_words = []

    # Get vector for each word in vocabulary if in model
    for idx, word in enumerate(vocab_words):
        if word in model:
            vocab_tensors[idx] = model.query(word)
        else:
            unk_words.append(word)

    # Override special tokens
    special_tokens = ['SOS', 'EOS', 'UNK']

    # Override special tokens
    vocab_tensors[:len(special_tokens), :] = np.random.uniform(
            -0.1, 0.1, (len(special_tokens), model.dim)).astype('float32')

    print('Tensor vocabulary complete.')
    print('    Total vocabulary size {}, {} UNK words ({:.2}%)'.format(len(vocab_words),
len(unk_words),
    (len(unk_words) / len(vocab_words)) *100))
    return torch.tensor(vocab_tensors, dtype=torch.float64), unk_words

class InitializedEncoderRNN(nn.Module):
    """Encoder class which trains embeddings from scratch and specifies GRU architecture"""
    def __init__(self, input_size, embedding_size, hidden_size, caption_vocab_index, freeze_weights):
        super(InitializedEncoderRNN, self).__init__()
        self.name = 'InitializedEncoderRNN'
        self.trainable_model = True
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.freeze_weights = freeze_weights
```

```
        self.embedding = nn.Embedding(input_size, self.embedding_size)
        self.embedding.weight = nn.Parameter(create_vocab_tensors(caption_vocab_index)[0])
        if freeze_weights == True:
            self.embedding.weight.requires_grad = False
        self.gru = nn.GRU(self.embedding_size, self.hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class GloveEncoder():
    """Encodes an input sentence as a mean or max pooled sentence embedding given the individual word
embeddings"""
    def __init__(self, pooling='mean'):
        self.name = 'GloveEncoder'
        self.trainable_model = False
        self.pooling = pooling
        self.model = Magnitude(config.glove_magnitude_path)
        self.hidden_size = self.model.dim

    def sentence_embedding(self, input_text):
        words_in_model = [word for word in input_text.split() if word in self.model]
        sentence_embedding = np.zeros((len(words_in_model), self.model.dim))
        sentence_embedding.fill(np.nan)

        for idx, token in enumerate(words_in_model):
            sentence_embedding[idx] = self.model.query(token)

        if self.pooling == 'max':
            sentence_embedding = np.max(sentence_embedding, axis=0)

        else:
            sentence_embedding = np.mean(sentence_embedding, axis=0)

        return torch.tensor(sentence_embedding.reshape(1, 1, -1), device=DEVICE)

def load_InferSent_model(vocab_size=250000, enable_GPU=True):
    """   Loads the pretrained InferSent model
    Based on https://github.com/facebookresearch/InferSent,
    note: needed to download file manually from:
https://dl.fbaipublicfiles.com/senteval/infersent/infersent1.pkl
    and also make changes to data and model files per:
https://github.com/facebookresearch/InferSent/issues/98 """

    model_version = 1 # Uses Glove Embeddings
    MODEL_PATH = config.infersent_model_path
    params_model = {'bsize': 64, 'word_emb_dim': 300, 'enc_lstm_dim': 2048,
                    'pool_type': 'max', 'dpout_model': 0.0, 'version': model_version}
    model = InferSent(params_model)
    model.load_state_dict(torch.load(MODEL_PATH))

    # Put model on GPU
    if enable_GPU:
        model = model.cuda()
    else:
        model

    model.set_w2v_path(config.glove_txt_path)

    # Load embeddings of K most frequent words
    model.build_vocab_k_words(K=vocab_size)
    return model

class InferSentEncoder():
    """Class designed for converting an input sentence to an embedding using InferSent"""
    def __init__(self):
        self.name = 'InferSentEncoder'
        self.trainable_model = False
        self.model = load_InferSent_model(vocab_size=250000, enable_GPU=GPU_ENABLED)
        self.hidden_size = 4096
```

```
    def sentence_embedding(self, input_text):
        embedded_sentence = self.model.encode([input_text], bsize=128, tokenize=False, verbose=False)
        return torch.tensor(embedded_sentence, device=DEVICE).view(1, 1, -1)

class BERTEncoder():
    """Class designed for converting an input sentence to an embedding using fine-tuned BERT"""
    def __init__(self):
        self.name = 'BERTEncoder'
        self.trainable_model = False
        self.model = SentenceTransformer('bert-base-nli-mean-tokens')
        self.hidden_size = 768

    def sentence_embedding(self, input_text):
        #https://huggingface.co/pytorch-transformers/model_doc/bert.html#bertmodel
        with torch.no_grad():
            encoded_sentence = self.model.encode([input_text], batch_size=1,
show_progress_bar=False)[0]
            return torch.tensor(encoded_sentence, device=DEVICE).view(1,1,-1)

class GPTLanguageModel():
    """Class designed for returning the fluency score using GPT for an input sentence"""
    def __init__(self):
        self.name = 'GPTLanguageModel'
        self.trainable_model = False
        self.GPT_tokenizer = OpenAIGPTTokenizer.from_pretrained('openai-gpt')
        self.model = OpenAIGPTLMHeadModel.from_pretrained('openai-gpt').eval()

    def fluency_score(self, input_sentence, max_value=500):
        try:
            with torch.no_grad():
                tokenize_input = self.GPT_tokenizer.encode(input_sentence)
                input_ids = torch.tensor(tokenize_input).unsqueeze(0)
                loss=self.model(input_ids, labels=input_ids)[0]
            return 1-min((math.exp(loss)/max_value), 0.99)
        except:
            print('GPT rejected sentence: ', input_sentence)
            return 0.01

class GPT2LanguageModel():
    """Class designed for returning the fluency score using GPT2 for an input sentence"""
    def __init__(self):
        self.name = 'GPT2LanguageModel'
        self.trainable_model = False
        self.GPT2_tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
        self.model = GPT2LMHeadModel.from_pretrained('gpt2').eval()

    def fluency_score(self, input_sentence, max_value=500):
        try:
            with torch.no_grad():
                tokenize_input = self.GPT2_tokenizer.encode(input_sentence)
                input_ids = torch.tensor(tokenize_input).unsqueeze(0)
                loss=self.model(input_ids, labels=input_ids)[0]
            return 1-min((math.exp(loss)/max_value), 0.99)
        except:
            print('GPT rejected sentence: ', input_sentence)
            return 0.01

#%% ---------------------- ARCHIVE ---------------------

#class BERTEncoder():
#    """$$$ general BERT without finetuning"""
#    def __init__(self):
#        self.name = 'BERTEncoder'
#        self.trainable_model = False
#        self.BERT_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
#        self.model = BertModel.from_pretrained('bert-base-uncased')
#        self.hidden_size = 768
#
#    def sentence_embedding(self, input_text):
#        #https://huggingface.co/pytorch-transformers/model_doc/bert.html#bertmodel
#        with torch.no_grad():
#            tokenized_text = self.BERT_tokenizer.encode(input_text)
#            input_ids = torch.tensor(tokenized_text).unsqueeze(0)  # Batch size 1
#
#            outputs = self.model(input_ids)
#
```

```
#            # Uses max pooling instead of classification token as apparently
#            # classification token does not contain meaningful semantic information
#            return torch.max(outputs[0], 1)[0]
```

9.3 Supervised Model

```
"""Defines, trains, and evaluates the defined supervised model with MLE.
    Includes modifications for teacher forcing and attention."""

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F

import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import argparse

import time
import math
import regex as re
import operator
from queue import PriorityQueue
from collections import deque

import os

import config
import data
import model_evaluation
import encoder_models
import utils

# Configure hyperparameters
DEVICE = config.DEVICE
GPU_ENABLED = config.GPU_ENABLED

MAX_LENGTH = config.MAX_LENGTH
SOS_token = config.SOS_token
EOS_token = config.EOS_token
UNK_token = config.UNK_token

#Load data
train_pairs = data.TRAIN_PAIRS
val_pairs = data.VAL_PAIRS
test_pairs = data.TEST_PAIRS
vocab_index = data.VOCAB_INDEX

# Set experiment args from command line
parser = argparse.ArgumentParser(description='Train_Supervised_Model')
parser.add_argument('--train_models', action='store_true',
                    help='enable training of models')
parser.add_argument('--folder_name', type=str,
                    help='Brief description of experiment (no spaces)')

parser.add_argument('--n_iterations', type=int, default=15,
                    help='number of iterations to run the training loop (default: 15)')
parser.add_argument('--n_epochs', type=int, default=5000,
                    help='number of epochs to train online in each loop (default: 5000)')
parser.add_argument('--early_stoppage_holdoff', type=int, default=5,
                    help='number of iterations where validation loss can be higher than min (default:
2)')

parser.add_argument('--start_tf_ratio', type=float, default=0.90,
                    help='ratio of teacher forcing at start (default: 0.90)')
parser.add_argument('--end_tf_ratio', type=float, default=0.85,
                    help='ratio of teacher forcing at end (default: 0.85)')
parser.add_argument('--tf_decay_iters', type=int, default=5,
                    help='number of iterations it takes for teacher forcing to decay to end value
(default: 5)')

parser.add_argument('--encoder_model', type=str,
                    choices=['VanillaEncoder']+encoder_models.pretrained_models_list,
```

```
                    default='VanillaEncoder', help='select encoder model (default: VanillaEncoder)')
parser.add_argument('--decoder_model', type=str,
                    choices=['VanillaDecoder', 'AttnDecoder'],
                    default='VanillaDecoder', help='select decoder model (default: VanillaEncoder)')
parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'Switch'], default='SGD',
                    help='optimizer used in training (default: SGD)')

parser.add_argument('--hidden_size', type=int, default=256,
                    help='number of hidden nodes in encoder and decoder (default: 256)')
parser.add_argument('--embedding_size', type=int, default=256,
                    help='number of hidden nodes in encoder and decoder embedding layers (default:
256)')

parser.add_argument('--load_models', action='store_true', help='Load pretrained model from prior
point')
parser.add_argument('--load_model_folder_name', type=str,
                    help='folder which contains the saved models to be used')

if __name__ == '__main__':
    args = parser.parse_args()
    args.save_models = 0

    if args.train_models:
        args.save_models = 1
        saved_supervised_model_results = data.SaveSupervisedModelResults(args.folder_name)
        saved_supervised_model_results.check_folder_exists()

#%% Manual Testing - turn train models on while keeping save models off then modify as you like

#args.train_models = 1
#saved_supervised_model_results = data.SaveSupervisedModelResults('test')
#
#args.encoder_model = 'VanillaEncoder'
#args.decoder_model = 'AttnDecoder'
#
#args.n_iterations = 3
#args.n_epochs = 100

#%% Define models and training / evaluation functions

class DecoderRNN(nn.Module):
    """Vanilla decoder which decodes based on single context vector"""
    def __init__(self, embedding_size, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.name = 'VanillaDecoderRNN'
        self.uses_attention = False
        self.is_agent = False
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, self.embedding_size)
        self.gru = nn.GRU(self.embedding_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class AttnDecoderRNN(nn.Module):
    """Attention decoder which decodes based on trained weightings over INPUT word context vectors.
    Does not currently attend over generated text while decoding"""
    def __init__(self, embedding_size, hidden_size, output_size, dropout_p=0.1,
max_length=MAX_LENGTH+1):
        super(AttnDecoderRNN, self).__init__()
        self.name = 'AttentionDecoderRNN'
        self.uses_attention = True
        self.is_agent = False
        self.hidden_size = hidden_size
        self.embedding_size = embedding_size
```

```
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.embedding_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.gru(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class Optimizers(nn.Module):
    """Defines optimizer object used in training,
        particularly relevant when using optimizer switching"""
    def __init__(self, encoder, decoder, switch_thresh=0.05):
        super(Optimizers, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        if encoder.trainable_model:
            self.encoder_optimizer = optim.Adam(encoder.parameters())
            self.encoder_opt_name = 'Adam'
        else:
            self.encoder_optimizer = 'Null'
            self.encoder_opt_name = 'Null'
        self.decoder_optimizer = optim.Adam(decoder.parameters())
        self.decoder_opt_name = 'Adam'
        self.training_loss = deque(maxlen=5)
        self.switch_thresh = switch_thresh
        self.enable_switch = True

    def optimizer_switch(self, force_switch_opt=False):
        """Designed to switch between Adam and SGD optimizers after initial reduction in performance"""
        # Change the optimizer to SGD if the difference between the min and mean is below threshold
        threshold_condition = False
        if len(self.training_loss) == self.training_loss.maxlen:
            threshold = (np.min(self.training_loss) / np.mean(self.training_loss)) -1
            threshold_condition = threshold < self.switch_thresh

        if threshold_condition or force_switch_opt:
            if self.encoder.trainable_model:
                self.encoder_optimizer = optim.SGD(self.encoder.parameters(), lr=0.01)
                self.encoder_opt_name = 'SGD'
            self.decoder_optimizer = optim.SGD(self.decoder.parameters(), lr=0.01)
            self.decoder_opt_name = 'SGD'
            self.enable_switch = False

class Teacher_Forcing_Ratio(object):
    """Defines the teacher forcing ratio which decays when update method is called"""
    def __init__(self, start_teacher_forcing_ratio, end_teacher_forcing_ratio, n_iterations):
        self.start_teacher_forcing_ratio = start_teacher_forcing_ratio
        self.end_teacher_forcing_ratio = end_teacher_forcing_ratio
        self.teacher_decay = (self.start_teacher_forcing_ratio - self.end_teacher_forcing_ratio) /
n_iterations
        self.teacher_forcing_ratio = self.start_teacher_forcing_ratio
```

```
    def update_teacher_forcing_ratio(self):
        self.teacher_forcing_ratio = np.max([self.teacher_forcing_ratio - self.teacher_decay,
                               self.end_teacher_forcing_ratio])

class BeamSearchNode(object):
    """Beam node used in beam decoding implementation"""
    """ Notebook source: https://github.com/budzianowski/PyTorch-Beam-Search-
Decoding/blob/master/decode_beam.py """
    def __init__(self, hiddenstate, previousNode, wordId, logProb, length):
        self.h = hiddenstate
        self.prevNode = previousNode
        self.wordid = wordId
        self.logp = logProb
        self.leng = length

    def eval(self, alpha=1.0):
        reward = 0
        # Add here a function for shaping a reward

        return self.logp / float(self.leng - 1 + 1e-6) + alpha * reward

def beam_decode(input_pair, encoder, decoder, beam_width=5, n_output_sentences=1,
encoder_outputs=None):
    """Implements beam search decoding using specified encoder, decoder, and beam length"""
    """ Notebook source: https://github.com/budzianowski/PyTorch-Beam-Search-
Decoding/blob/master/decode_beam.py """
    '''
    :param target_tensor: target indexes tensor of shape [B, T] where B is the batch size and
    T is the maximum length of the output sentence
    :param decoder_hidden: input tensor of shape [1, B, H] for start of the decoding
    :param encoder_outputs: if you are using attention mechanism you can pass encoder outputs,
    [T, B, H] where T is the maximum length of input sentence
    :return: decoded_batch
    '''
    assert beam_width > 1, 'Beam width must be greater than 1'

    if encoder.trainable_model:
        input_tensor, _ = utils.tensorsFromPair(input_pair)

        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden()
        encoder_outputs = torch.zeros(MAX_LENGTH+1, encoder.hidden_size, device=DEVICE)

        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                     encoder_hidden)
            encoder_outputs[ei] += encoder_output[0, 0]

        decoder_hidden = encoder_hidden

    else:
        decoder_hidden = encoder.sentence_embedding(input_pair[0])
        decoder_hidden = layer_normalize(decoder_hidden)

    topk = n_output_sentences  # how many sentence do you want to generate

    # Start with the start of the sentence token
    decoder_input = torch.tensor([[SOS_token]], device=DEVICE)

    # Number of sentence to generate
    endnodes = []
    number_required = min((topk + 1), topk - len(endnodes))

    # starting node -  hidden vector, previous node, word id, logp, length
    node = BeamSearchNode(decoder_hidden, None, decoder_input, 0, 1)
    nodes = PriorityQueue()

    # start the queue
    nodes.put((-node.eval(), node))
    qsize = 1

    # start beam search
    for _ in range(2000):
        # give up when decoding takes too long
        if qsize > 1000: break
```

```
            # fetch the best node
            score, n = nodes.get()
            decoder_input = n.wordid
            decoder_hidden = n.h

            if n.wordid.item() == EOS_token and n.prevNode != None:
                endnodes.append((score, n))
                # if we reached maximum # of sentences required
                if len(endnodes) >= number_required:
                    break
                else:
                    continue

            # decode for one step using decoder
            if decoder.uses_attention:
                decoder_output, decoder_hidden, _ = decoder(
                    decoder_input, decoder_hidden, encoder_outputs)
            else:
                decoder_output, decoder_hidden = decoder(
                    decoder_input, decoder_hidden)

            # do actual beam search
            log_prob, indexes = torch.topk(decoder_output, beam_width)
            nextnodes = []

            for new_k in range(beam_width):
                decoded_t = indexes[0][new_k].view(1, -1)
                log_p = log_prob[0][new_k].item()

                node = BeamSearchNode(decoder_hidden, n, decoded_t, n.logp + log_p, n.leng + 1)
                score = -node.eval()
                nextnodes.append((score, node))

            # put them into queue
            for i in range(len(nextnodes)):
                score, next_node = nextnodes[i]
                nodes.put((score, next_node))
                # increase qsize
            qsize += len(nextnodes) - 1

        # choose nbest paths, back trace them
        if len(endnodes) == 0:
            endnodes = [nodes.get() for _ in range(topk)]

        utterances = []
        for score, n in sorted(endnodes, key=operator.itemgetter(0)):
            utterance = []
            utterance.append(n.wordid)
            # back trace
            while n.prevNode != None:
                n = n.prevNode
                utterance.append(n.wordid)

            utterance = utterance[::-1]
            utterances.append(utterance)

        output_sentences = []
        for sentence in utterances:
            output_words = [vocab_index.index2word[word_idx.item()] for word_idx in sentence]
            output_sentences.append(' '.join(output_words[1:-1]))

        return output_sentences

    def asMinutes(s):
        """Converts time to minutes for print output"""
        m = math.floor(s / 60)
        s -= m * 60
        return '%dm %ds' % (m, s)

    def timeSince(since, percent):
        """Tracks time change for print output"""
        now = time.time()
        s = now - since
        es = s / (percent)
        rs = es - s
        return '%s (- %s)' % (asMinutes(s), asMinutes(rs))
```

```
def showAttention(input_sentence, output_words, attentions):
    """Helper function which creates graph for plotting attention alignments table"""
    # Set up figure with colorbar
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(attentions.numpy(), cmap='bone')
    fig.colorbar(cax)

    # Set up axes
    ax.set_xticklabels([''] + input_sentence.split(' ') +
                       ['<EOS>'], rotation=90)
    ax.set_yticklabels([''] + output_words)

    # Show label at every tick
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()

def evaluateAndShowAttention(input_pair, encoder, decoder):
    """Creates output which shows the attention alignments between the generated text and input"""
    output_words, attentions = evaluate(input_pair, encoder, decoder,
                                       max_length=MAX_LENGTH)
    print('input =', input_pair[0])
    print('output =', ' '.join(output_words))
    showAttention(input_pair[0], output_words, attentions)

def n_model_parameters(model):
    """Returns the number of trainable parameters for a given model"""
    model_parameters = filter(lambda p: p.requires_grad, model.parameters())
    params = sum([np.prod(p.size()) for p in model_parameters])
    return params

class LayerNorm(nn.Module):
    """Handles layer normalization used to normalize features between RNN layers"""
    """ Source: https://github.com/pytorch/pytorch/issues/1959"""
    def __init__(self, features, eps=1e-6):
        super().__init__()
        self.gamma = nn.Parameter(torch.ones(features))
        self.beta = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.gamma * (x - mean) / (std + self.eps) + self.beta

def layer_normalize(input_features, output_shape = (1,1,-1)):
    """Applies layer normalization used to normalize features between RNN layers"""
    # Initialize layernorm object
    layer_norm = LayerNorm(input_features.squeeze().shape).to(DEVICE)

    # Normalize features and reshape
    normalized_features = layer_norm(input_features.squeeze().float())
    normalized_features = normalized_features.view(output_shape).detach()
    return normalized_features

def train_encoder(input_tensor, encoder, encoder_optimizer, max_length=MAX_LENGTH):
    """Initializes encoder model and generates encoder preds for model training"""
    # Initialize empty hidden layer
    encoder_hidden = encoder.initHidden()

    # Clear the gradients from the optimizers
    encoder_optimizer.zero_grad()

    # Initialize arrays for vector length of input and target
    input_length = input_tensor.size(0)

    # Initialize encoder outputs and instantiate the loss (includes padding room)
    encoder_outputs = torch.zeros(max_length+1, encoder.hidden_size, device=DEVICE)

    # Encodes input tensor into [len x hidden layer] sentence embedding matrix
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
```

```
        encoder_outputs[ei] = encoder_output[0, 0]

    return encoder_hidden, encoder_outputs, encoder_optimizer

#%%
def train(input_pair, encoder, decoder, encoder_optimizer, decoder_optimizer,
          criterion, teacher_forcing_ratio, max_length=MAX_LENGTH):
    """Model training logic, initializes graph, creates encoder outputs matrix for attention model,
    applies teacher forcing (randomly), calculates the loss and trains the models"""
    if encoder.trainable_model:
        # Encode sentences using encoder model
        input_tensor, target_tensor = utils.tensorsFromPair(input_pair)
        decoder_hidden, encoder_outputs, encoder_optimizer = train_encoder(
                    input_tensor, encoder, encoder_optimizer, max_length)
    else:
        # Encode sentences using pretrained encoder model
        target_tensor = utils.tensorFromSentence(vocab_index, input_pair[1])
        decoder_hidden = encoder.sentence_embedding(input_pair[0])
        decoder_hidden = layer_normalize(decoder_hidden)

    # Clear the gradients from the decoder optimizer
    decoder_optimizer.zero_grad()
    target_length = target_tensor.size(0)

    decoder_input = torch.tensor([[SOS_token]], device=DEVICE)
    loss = 0

    # Randomly apply teacher forcing subject to teacher forcing ratio
    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            if decoder.uses_attention:
                decoder_output, decoder_hidden, _ = decoder(
                    decoder_input, decoder_hidden, encoder_outputs)
            else:
                decoder_output, decoder_hidden = decoder(
                    decoder_input, decoder_hidden)

            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing: set next input to correct target

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            if decoder.uses_attention:
                decoder_output, decoder_hidden, _ = decoder(
                    decoder_input, decoder_hidden, encoder_outputs)
            else:
                decoder_output, decoder_hidden = decoder(
                    decoder_input, decoder_hidden)

            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break

    # Calculate the error and blackpropogate through the network
    loss.backward()

    if encoder.trainable_model:
        encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length

def trainIters(input_pairs, encoder, decoder, encoder_optimizer, decoder_optimizer,
               n_iters=5000, print_every=1000, teacher_forcing_ratio=0.9):
    """Training loop including setting optimizers and loss function, currently trains
    online using each example instead of batches"""
    start = time.time()
```

```
    print_loss_total = 0  # Reset every print_every

    criterion = nn.NLLLoss()

    # Sample n random pairs
    training_pairs = data.sample_list(input_pairs, n_iters)

    # For EACH pair train model to decrease loss
    for idx, pair in enumerate(training_pairs):
        loss = train(pair, encoder, decoder, encoder_optimizer,
                     decoder_optimizer, criterion, teacher_forcing_ratio)
        print_loss_total += loss

        iter = idx+1
        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            if opt.enable_switch:
                opt.training_loss.append(print_loss_avg)
            if args.save_models:
                saved_supervised_model_results.train_loss.append(np.around(print_loss_avg,2))
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100, print_loss_avg))

def embed_input_sentence(input_pair, encoder, max_length=MAX_LENGTH):
    """Embeds the input sentence using a trained encoder model"""
    with torch.no_grad():
        if encoder.trainable_model:
            input_tensor, target_tensor = utils.tensorsFromPair(input_pair)

            input_length = input_tensor.size()[0]
            encoder_hidden = encoder.initHidden()
            encoder_outputs = torch.zeros(max_length+1, encoder.hidden_size, device=DEVICE)

            for ei in range(input_length):
                encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                         encoder_hidden)
                encoder_outputs[ei] += encoder_output[0, 0]

            decoder_hidden = encoder_hidden

            return decoder_hidden, target_tensor, encoder_outputs

        else:
            target_tensor = utils.tensorFromSentence(vocab_index, input_pair[1])
            decoder_hidden = encoder.sentence_embedding(input_pair[0])
            decoder_hidden = layer_normalize(decoder_hidden)

            return decoder_hidden, target_tensor, None

def evaluate(input_pair, encoder, decoder, max_length=MAX_LENGTH):
    """Generates the supervised prediction for a given input sentence"""
    if encoder.trainable_model:
        decoder_hidden, target_tensor, encoder_outputs = embed_input_sentence(input_pair, encoder,
max_length)
    else:
        decoder_hidden, target_tensor, _ = embed_input_sentence(input_pair, encoder, max_length)

    with torch.no_grad():
        decoder_input = torch.tensor([[SOS_token]], device=DEVICE)  # SOS

        decoded_words = []
        decoder_attentions = torch.zeros(max_length+1, max_length+1)

        for di in range(max_length-1):
            if decoder.uses_attention:
                decoder_output, decoder_hidden, decoder_attention = decoder(
                    decoder_input, decoder_hidden, encoder_outputs)
                decoder_attentions[di] = decoder_attention.data
            else:
                decoder_output, decoder_hidden = decoder(
                    decoder_input, decoder_hidden)

            topv, topi = decoder_output.data.topk(1)

            decoder_input = topi.squeeze().detach()  # AL # detach from history as input
```

```
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(vocab_index.index2word[topi.item()])

            decoder_input = topi.squeeze().detach()

        if decoder.uses_attention:
            return decoded_words, decoder_attentions[:di + 1]
        else:
            return decoded_words, 'None'

def evaluateError(input_pair, encoder, decoder, max_length=MAX_LENGTH):
    """Generates the predictions as well as the error using teacher forcing"""
    criterion = nn.NLLLoss()

    if encoder.trainable_model:
        decoder_hidden, target_tensor, encoder_outputs = embed_input_sentence(input_pair, encoder,
max_length)
    else:
        decoder_hidden, target_tensor, _ = embed_input_sentence(input_pair, encoder, max_length)

    target_length = target_tensor.size(0)
    with torch.no_grad():
        decoder_input = torch.tensor([[SOS_token]], device=DEVICE)  # SOS

        decoded_words = []
        loss = 0

        use_teacher_forcing = True if random.random() < tf_ratio.teacher_forcing_ratio else False

        if use_teacher_forcing:
            for di in range(target_length):
                if decoder.uses_attention:
                        decoder_output, decoder_hidden, _ = decoder(
                            decoder_input, decoder_hidden, encoder_outputs)
                else:
                    decoder_output, decoder_hidden = decoder(
                            decoder_input, decoder_hidden)
                loss += criterion(decoder_output, target_tensor[di]) # AL
                decoder_input = target_tensor[di]
                decoded_words.append(vocab_index.index2word[target_tensor[di].item()])
        else:
            for di in range(max_length):
                if decoder.uses_attention:
                    decoder_output, decoder_hidden, _ = decoder(
                        decoder_input, decoder_hidden, encoder_outputs)
                else:
                    decoder_output, decoder_hidden = decoder(
                        decoder_input, decoder_hidden)

                topv, topi = decoder_output.data.topk(1)

                decoder_input = topi.squeeze().detach()  # AL # detach from history as input

                if di < len(target_tensor):
                    loss += criterion(decoder_output, target_tensor[di]) # AL

                if topi.item() == EOS_token:
                    decoded_words.append('<EOS>')
                    break
                else:
                    decoded_words.append(vocab_index.index2word[topi.item()])

        return decoded_words, loss.item() / target_length

def generateSentences(input_pairs, encoder, decoder, n=10):
    """Generates sentences based on encoder and decoder models given an input"""
    for i in range(n):
        pair = random.choice(input_pairs)
        print('>', pair[0])
        print('=', pair[1])

        output_words = evaluate(pair, encoder, decoder)[0]
```

```
        output_sentence = ' '.join(output_words)
        print('<', output_sentence)
        print('')

def validationError(input_pairs, encoder, decoder, verbose=True):
    """Evalutes the error on a set of input pairs in terms of loss.
    Is intended to be used on a validation or test set to evaluate performance"""
    loss = 0

    for pair in input_pairs:
        _, pair_loss = evaluateError(pair, encoder, decoder)
        loss += pair_loss

    avg_loss = loss / len(input_pairs)

    if verbose:
        print('The average validation loss is {:.3} based on {} samples'.format(avg_loss,
len(input_pairs)))
    return avg_loss

def predict_sentences(input_pairs, encoder, decoder):
    """Predicts the generated outputs for a set of input sentences"""
    pred_pairs = []

    for pair in input_pairs:
        output_words = evaluate(pair, encoder, decoder)[0]
        output_sentence = ' '.join(output_words)
        output_sentence = re.sub(r" <EOS>", r"", output_sentence)
        pred_pairs.append([pair[1], output_sentence])
    return np.array(pred_pairs)

def validationMetricPerformance(input_pairs, encoder, decoder, similarity_model=None,
fluency_model=None,
                                ESIM_model=None, logr_model=None, std_scaler=None,
                                similarity_dist=None, fluency_dist=None, ESIM_dist=None,
                                vocab_index=vocab_index, verbose=True, metric='BLEU1'):
    """Returns the model performance on a specified reward metric"""
    pred_pairs = predict_sentences(input_pairs, encoder, decoder)

    metrics_perf = np.array([model_evaluation.performance_metrics(
            target_sent, pred_sent, similarity_model=similarity_model, fluency_model=fluency_model,
            ESIM_model=ESIM_model, logr_model=logr_model, std_scaler=std_scaler,
            similarity_dist=similarity_dist, fluency_dist=fluency_dist,
            ESIM_dist=ESIM_dist, vocab_index=vocab_index,
            metric=metric) for (target_sent, pred_sent) in pred_pairs])

    mean_performance = metrics_perf.mean()

    if verbose:
        print('Average metric performance: {}'.format(mean_performance))

    return pred_pairs, metrics_perf, mean_performance

def model_pipeline(n_iterations, encoder, decoder, encoder_optimizer, decoder_optimizer,
n_epochs=5000):
    """Model pipeline which trains model and also generates examples while training and evaluation
    on the validation set for potential early stopping. The teacher forcing ratio can also be
    adjusted stepwise as desired"""

    for i in range(n_iterations):
        print('-------------------- Iteration {}: --------------------'.format(i+1))
        print('    Teacher Forcing Ratio: {:.2} | Optimizer: {}'.format(
                tf_ratio.teacher_forcing_ratio, opt.decoder_opt_name))
        if opt.enable_switch:
            opt.optimizer_switch()
        trainIters(train_pairs, encoder, decoder, encoder_optimizer, decoder_optimizer, n_epochs,
                   print_every=1000, teacher_forcing_ratio=tf_ratio.teacher_forcing_ratio)
        generateSentences(val_pairs, encoder, decoder, n=3)
        avg_val_loss = validationError(data.sample_list(val_pairs, sample_by_prop=True,
sample_prop=0.20),
                                       encoder, decoder)
        tf_ratio.update_teacher_forcing_ratio()

        saved_supervised_model_results.val_loss.append(np.around(avg_val_loss, 2))

        if args.save_models:
```

```
            saved_supervised_model_results.export_loss('training_loss.txt', 'val_loss.txt')

            if avg_val_loss <= saved_supervised_model_results.val_loss_thresh:
                if encoder.trainable_model:
                    saved_supervised_model_results.save_top_models(encoder,
'encoder_{:.3f}.pt'.format(avg_val_loss))
                    saved_supervised_model_results.save_top_models(decoder,
'decoder_{:.3f}.pt'.format(avg_val_loss))
                else:
                    saved_supervised_model_results.save_top_models(decoder,
'decoder_{:.3f}.pt'.format(avg_val_loss))

        if (len(saved_supervised_model_results.val_loss[:-1]) > 0) and \
        (avg_val_loss > min(saved_supervised_model_results.val_loss[:-1])):
            args.early_stoppage_holdoff -= 1
            if args.early_stoppage_holdoff <= 0:
                break

def define_encoder(encoder_name='VanillaEncoder'):
    """Initializes the encoder model"""
    # Instantiate encoder and decoder models
    if encoder_name == 'VanillaEncoder':
        return encoder_models.EncoderRNN(input_size=vocab_index.n_words,
                                       embedding_size=args.embedding_size,
                                       hidden_size=args.hidden_size).to(DEVICE)

    elif encoder_name == 'InitializedEncoder':
        return encoder_models.InitializedEncoderRNN(
                input_size=vocab_index.n_words, embedding_size=300, hidden_size=args.hidden_size,
                caption_vocab_index=vocab_index, freeze_weights=True).to(DEVICE)

    elif encoder_name == 'GloveEncoder':
        return encoder_models.GloveEncoder()

    elif encoder_name == 'InferSentEncoder':
        return encoder_models.InferSentEncoder()

    elif encoder_name == 'BERTEncoder':
        return encoder_models.BERTEncoder()

    else:
        print("Please specify one of the following encoders: {}".format(
                ['VanillaEncoder']+encoder_models.pretrained_models_list))

def define_decoder(encoder, decoder_name='VanillaDecoder'):
    """Initializes the decoder model"""
    hidden_size = encoder.hidden_size
    decoder_embedding_size = args.embedding_size

    if decoder_name == 'VanillaDecoder':
        return DecoderRNN(embedding_size=decoder_embedding_size, hidden_size=hidden_size,
                          output_size=vocab_index.n_words).to(DEVICE)
    elif decoder_name == 'AttnDecoder':
        return AttnDecoderRNN(embedding_size=decoder_embedding_size, hidden_size=hidden_size,
                              output_size=vocab_index.n_words, dropout_p=0.1).to(DEVICE)
    else:
        print("""Please specify one of the following decoders:
            ['VanillaDecoder', 'AttnDecoder']""")

def load_supervised_models(folder_name, encoder_file_name='best', decoder_file_name='best'):
    """Initializes the encoder and decoder models and loads the weights from the trained models"""
    hidden_size=256; embedding_size=256;
    if encoder_file_name in encoder_models.pretrained_models_list:
        encoder = define_encoder(encoder_file_name)
        decoder = DecoderRNN(embedding_size=embedding_size, hidden_size=encoder.hidden_size,
                             output_size=vocab_index.n_words).to(DEVICE)
        if decoder_file_name == 'best':
            decoder_file_name = 'decoder_{:.3f}.pt'.format(data.get_top_n_models(
                    os.path.join(config.saved_supervised_model_path, folder_name), 'decoder', n=1,
descending=False)[0])

        data.load_model(decoder, os.path.join(config.saved_supervised_model_path, folder_name,
decoder_file_name))
        return encoder, decoder

    else:
```

```
        if folder_name in ['Baseline_Test', 'VanillaEncoder', 'VanillaEncoder_Adam',
                           'VanillaEncoder_Switch', 'VanillaEncoder_TF']:
            pass

        elif folder_name in ['Attention_SGD', 'Attention_Adam']:
            decoder = AttnDecoderRNN(embedding_size=embedding_size, hidden_size=hidden_size,
                                     output_size=vocab_index.n_words).to(DEVICE)

        else:
            raise SystemExit('Please correct test folder name')

        if 'encoder' not in locals():
            encoder = encoder_models.EncoderRNN(input_size=vocab_index.n_words,
                                               embedding_size=embedding_size,
hidden_size=hidden_size).to(DEVICE)
        if 'decoder' not in locals():
            decoder = DecoderRNN(embedding_size=embedding_size, hidden_size=hidden_size,
                                 output_size=vocab_index.n_words).to(DEVICE)

        if encoder_file_name == 'best':
            encoder_file_name = 'encoder_{:.3f}.pt'.format(data.get_top_n_models(
                    os.path.join(config.saved_supervised_model_path, folder_name), 'encoder', n=1,
descending=False)[0])

        if decoder_file_name == 'best':
            decoder_file_name = 'decoder_{:.3f}.pt'.format(data.get_top_n_models(
                    os.path.join(config.saved_supervised_model_path, folder_name), 'decoder', n=1,
descending=False)[0])

        data.load_model(encoder, os.path.join(config.saved_supervised_model_path, folder_name,
encoder_file_name))
        data.load_model(decoder, os.path.join(config.saved_supervised_model_path, folder_name,
decoder_file_name))

        return encoder, decoder

def test_sentence_evaluation(input_pair, encoder, decoder):
    """Produces a predicted sentence using both MLE and beam search"""
    print('Input Sentence: ', input_pair[0])
    print('Target Sentence: ', input_pair[1])
    print()
    print('MLE Model Output:',' '.join(evaluate(input_pair, encoder, decoder)[0]))
    print('Beam Search Model Output:', beam_decode(input_pair=input_pair, encoder=encoder,
decoder=decoder,
                beam_width=4, n_output_sentences=1, encoder_outputs=None))

#    if decoder.uses_attention:
#        evaluateAndShowAttention(input_pair, encoder, decoder)

#%% Train and Evaluate Model

if (__name__ == '__main__') and args.train_models:
    """Initializes models subject to cmd line args and then trains and evaluates performance"""

    # Initialize encoder and decoder models
    encoder = define_encoder(encoder_name=args.encoder_model)
    decoder = define_decoder(encoder, decoder_name=args.decoder_model)

    # Load saved weights from specified model
    if args.load_models:
        if args.encoder_model in encoder_models.pretrained_models_list:
            encoder, decoder = load_supervised_models(
                    args.load_model_folder_name, encoder_file_name=args.encoder_model,
                    decoder_file_name='best')
        else:
            encoder, decoder = load_supervised_models(
                    args.load_model_folder_name, encoder_file_name='best',
                    decoder_file_name='best')

    # Initialize folder if saving models
    if args.save_models:
        saved_supervised_model_results.init_folder(args, encoder, decoder)


    # Specify numberr of iterations and the optimizer used
    n_iterations = args.n_iterations
```

```
    opt = Optimizers(encoder, decoder, switch_thresh=0.05)
    if args.optimizer == 'SGD':
        opt.optimizer_switch(force_switch_opt=True)
    elif args.optimizer == 'Adam':
        opt.enable_switch = False
    else:
        pass

    # Initialize teacher forcing object as well as key args
    tf_ratio = Teacher_Forcing_Ratio(start_teacher_forcing_ratio=args.start_tf_ratio,
                                     end_teacher_forcing_ratio=args.end_tf_ratio,
                                     n_iterations=args.tf_decay_iters)

    # Train model subject to args and save if error
    try:
        model_pipeline(n_iterations=n_iterations, encoder=encoder, decoder=decoder,
                       encoder_optimizer=opt.encoder_optimizer,
decoder_optimizer=opt.decoder_optimizer,
                       n_epochs=args.n_epochs)
    except:
        if args.save_models:
            saved_supervised_model_results.export_loss('training_loss.txt', 'val_loss.txt')

            try:
                saved_supervised_model_results.save_top_models(encoder, 'encoder_CHECKPOINT.pt')
                saved_supervised_model_results.save_top_models(decoder, 'decoder_CHECKPOINT.pt')
            except:
                saved_supervised_model_results.save_top_models(decoder, 'decoder_CHECKPOINT.pt')
```

9.4 *Model Evaluation*

```
"""Contains wrappers for different evaluation functions to be used primarily as
    reward functions for RL model"""

import numpy as np

import os
import config
import utils
import data
from train_ESIM import ESIM_pred, load_ESIM_model
import encoder_models

#%%

from eval_metrics.bleu.bleu_scorer import BleuScorer
from eval_metrics.rouge.rouge import Rouge
from eval_metrics.cider.cider_scorer import CiderScorer
from eval_metrics.meteor.meteor import Meteor

# Define conventional automatic metrics warppers from eval_metrics package
def BLEU_score(target_sentence, pred_sentence, n_tokens=1):
    """Returns BLEU score at specified n-gram level for a given target and predicted sentence pair"""
    try:
        # Set n to BLEU score level
        bleu_scorer = BleuScorer(n=n_tokens)
        bleu_scorer += (pred_sentence[0], target_sentence)
        BLEU_score, _ = bleu_scorer.compute_score()
        return np.around(BLEU_score[n_tokens-1], 4)
    except:
        print('rejected sentence: ', pred_sentence)

def ROUGE_score(target_sentence, pred_sentence):
    """Returns ROUGE score for a given target and predicted sentence pair"""
    try:
        rouge = Rouge()
        ROUGE_score = rouge.calc_score(pred_sentence, target_sentence)
        return np.around(ROUGE_score, 4)
    except:
        print('rejected sentence: ', pred_sentence)

def CIDER_score(target_sentence, pred_sentence):
```

```
    """Returns CIDER score for a given target and predicted sentence pair"""
    try:
        cider_scorer = CiderScorer(idf_terms_path=os.path.join(config.PATH,
'eval_metrics/cider/idf_terms'))
        cider_scorer += (pred_sentence[0], target_sentence)
        CIDER_score, _ = cider_scorer.compute_score()
        return np.around(CIDER_score, 4)
    except:
        print('rejected sentence: ', pred_sentence)

def METEOR_score(target_sentence, pred_sentence):
    """Returns METEOR score for a given target and predicted sentence pair"""
    try:
        meteor = Meteor()
        METEOR_score, _ = meteor.compute_score(pred_sentence,
                            target_sentence)
        return np.around(METEOR_score, 4)
    except:
        print("Java did not execute properly.")

#%%

def tokenize(input_sentence):
    """Converts an input sentence to a set of tokens after applying preprocessing"""
    preprocessed_sentence = data.preprocess(input_sentence, remove_punct=True, lower_case=True)
    tokens = preprocessed_sentence.split()
    return tokens

def sentence_similarity(target, pred, similarity_model):
    """Calculates the cosine similarity between the sentence embeddings of a target and predicted pair
        using the embedding model specified"""
    try:
        cosine_sim = utils.cosine_similarity(similarity_model.sentence_embedding(target).view(1,-1),
                                       similarity_model.sentence_embedding(pred).view(1,-1))
        return np.around(cosine_sim.item(),4)
    except:
        print('similarity rejected sentence: ', pred)
        return 0.01

def sentence_length(input_sentence, min_value=6, max_value=12):
    """Returns the sentence length score for use as an auxiliary reward function"""
    input_value = np.clip(len(input_sentence.split()), min_value, max_value)
    return 1 - ((input_value - min_value) / (max_value - min_value))

def avg_word_frequency(input_sentence, vocab_index, total_word_count):
    """Returns the average word frequency for use as an auxiliary reward function"""
    return np.mean([vocab_index.word2count[word] for word in input_sentence.split()]) /
total_word_count

def rare_word_prop(input_sentence, vocab_index, rare_thresh=10):
    """Returns the proportion of rare words in a sentence for use as an auxiliary reward function"""
    try:
        assert len(input_sentence) > 0, 'Sentence is empty'
        input_words = input_sentence.split()
        n_rare_words = 0
        for word in input_words:
            if vocab_index.word2count[word] <= rare_thresh:
                n_rare_words += 1
            else:
                pass
        return n_rare_words / len(input_words)
    except:
        print('rejected sentence: ', input_sentence)
        return 0.01

def word_syllable_count(input_word):
    """Returns the syllable count for a word"""
    """source: https://stackoverflow.com/questions/46759492/syllable-count-in-python"""
    vowels = "aeiouyAEIOUY"
    count = 0
    prior_letter = None

    try:
        for idx, letter in enumerate(input_word):
            if (idx == 0) and (letter in vowels):
                count += 1
```

```
            elif (letter in vowels) and (prior_letter not in vowels):
                count += 1
            prior_letter = letter

        if input_word.endswith("e"):
            count -= 1
        return max(1, count)
    except:
        print('rejected word: ', input_word)
        return 1

def scaled_sent_syllable_count(input_sentence, max_avg_n_syllables=2):
    """Returns the average syllable count for a sentence for use as an auxiliary reward function"""
    try:
        assert len(input_sentence) > 0, 'Sentence is empty'
        avg_syllable_count = np.mean([word_syllable_count(word) for word in input_sentence.split()])
        return np.min([(avg_syllable_count / max_avg_n_syllables), 1])
    except:
        print('rejected sentence: ', input_sentence)
        return 0.01

def libertarian_pred(input_sentence, BERT_model, logr_model, std_scaler):
    """Returns the probability a given sentence is a comment from a Libertarian subreddit rather than
        an Anarchist / Socialist subreddit based on a trained model for use as an auxiliary reward
function"""
    try:
        encoded_sent = BERT_model.sentence_embedding(input_sentence).reshape(1, -1)
        standardized_sent = std_scaler.transform(encoded_sent)
        probs = logr_model.predict_proba(standardized_sent)
        return probs[0][1]

    except:
        print('rejected political sentence: ', input_sentence)
        return 0.01

def performance_metrics(target_sentence, pred_sentence, similarity_model=None, fluency_model=None,
                        ESIM_model=None, logr_model=None, std_scaler=None, similarity_dist=None,
fluency_dist=None,
                        ESIM_dist=None, vocab_index=None, metric='BLEU1'):
    """The main pipeline which handles applying the appropriate reward function and handles
        the relevant models / data inputs required"""
    if metric == 'BLEU1':
        return BLEU_score(target_sentence=[target_sentence], pred_sentence=[pred_sentence], n_tokens=1)

    if metric == 'BLEU2':
        return BLEU_score(target_sentence=[target_sentence], pred_sentence=[pred_sentence], n_tokens=2)

    elif metric == 'ROUGE':
        return ROUGE_score(target_sentence=[target_sentence], pred_sentence=[pred_sentence])

    elif metric == 'CIDER':
        return CIDER_score(target_sentence=[target_sentence], pred_sentence=[pred_sentence])

    elif metric == 'METEOR':
        return METEOR_score(target_sentence=[target_sentence], pred_sentence=[pred_sentence])

    elif metric == 'FLUENCY':
        return fluency_model.fluency_score(pred_sentence)

    elif metric == 'PARA':
        return sentence_similarity(target_sentence, pred_sentence, similarity_model)

    elif metric == 'PARA_F':
        similarity_score = sentence_similarity(target_sentence, pred_sentence, similarity_model)
        fluency_score = fluency_model.fluency_score(pred_sentence)

        scaled_similarity_score = utils.cdf_score(similarity_dist, similarity_score)
        scaled_fluency_score = utils.cdf_score(fluency_dist, fluency_score)

        return np.mean([scaled_similarity_score, scaled_fluency_score])

    elif metric == 'PARASIM':
        similarity_score = sentence_similarity(target_sentence, pred_sentence, similarity_model)
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()

        scaled_similarity_score = utils.cdf_score(similarity_dist, similarity_score)
```

```
        scaled_ESIM_score = utils.cdf_score(ESIM_dist, ESIM_score)

        return np.mean([scaled_similarity_score, scaled_ESIM_score])

    elif metric == 'PARASIM_F':
        similarity_score = sentence_similarity(target_sentence, pred_sentence, similarity_model)
        fluency_score = fluency_model.fluency_score(pred_sentence)
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()

        scaled_similarity_score = utils.cdf_score(similarity_dist, similarity_score)
        scaled_fluency_score = utils.cdf_score(fluency_dist, fluency_score)
        scaled_ESIM_score = utils.cdf_score(ESIM_dist, ESIM_score)

        return np.mean([scaled_similarity_score, scaled_fluency_score, scaled_ESIM_score])

    elif metric == 'ESIM':
        return ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()

    elif metric == 'ESIM_short':
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()
        length_score = sentence_length(pred_sentence)

        return 0.6 * ESIM_score + 0.4 * length_score

    elif metric == 'ESIM_syllables':
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()
        syllable_score = scaled_sent_syllable_count(pred_sentence, max_avg_n_syllables=2)

        return 0.6 * ESIM_score + 0.4 * syllable_score

    elif metric == 'ESIM_rare':
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()
        rare_score = rare_word_prop(pred_sentence, vocab_index, rare_thresh=2500)

        return 0.6 * ESIM_score + 0.4 * rare_score

    elif metric == 'ESIM_F':
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()
        fluency_score = fluency_model.fluency_score(pred_sentence)

        scaled_ESIM_score = utils.cdf_score(ESIM_dist, ESIM_score)
        scaled_fluency_score = utils.cdf_score(fluency_dist, fluency_score)

        return np.mean([scaled_ESIM_score, scaled_fluency_score])

    elif metric == 'ESIM_libertarian':
        ESIM_score = ESIM_pred([[target_sentence, pred_sentence]], ESIM_model, temperature=2).item()
        libertarian_score = libertarian_pred(pred_sentence, similarity_model, logr_model, std_scaler)

        return 0.6 * ESIM_score + 0.4 * libertarian_score

def init_eval_models(reward_function='BLEU1', similarity_model_name='BERT',
ESIM_model_name='ESIM_noisy_3'):
    """Initializes the appropriate models / data for use in the performance metrics evaluation.
        The following fields are being initialized:
        similarity_model, fluency_model, ESIM_model, \
        logr_model, std_scaler, \
        similarity_dist, fluency_dist, ESIM_dist"""
    if reward_function == 'FLUENCY':
        return None, encoder_models.GPTLanguageModel(), None, \
                            None, None, \
                            None, None, None

    if reward_function == 'PARA':
        if similarity_model_name =='BERT':
            return encoder_models.BERTEncoder(), None, None, \
                            None, None, \
                            None, None, None

        elif similarity_model_name =='InferSent':
            return encoder_models.InferSentEncoder(), None, None, \
                    None, None, \
                    None, None, None

    elif reward_function == 'PARA_F':
        if similarity_model_name =='BERT':
```

```
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'BERT_dist.npy'))
            fluency_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'fluency_dist.npy'))
            return encoder_models.BERTEncoder(), encoder_models.GPTLanguageModel(), None, \
                    None, None, \
                    similarity_dist, fluency_dist, None

        elif similarity_model_name =='InferSent':
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'InferSent_dist.npy'))
            fluency_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'fluency_dist.npy'))
            return encoder_models.InferSentEncoder(), encoder_models.GPTLanguageModel(), None, \
                    None, None, \
                    similarity_dist, fluency_dist, None

    elif reward_function == 'PARASIM':
        if similarity_model_name =='BERT':
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'BERT_dist.npy'))
            ESIM_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'ESIM_dist.npy'))
            return encoder_models.BERTEncoder(), None, load_ESIM_model(ESIM_model_name), \
                    None, None, \
                    similarity_dist, None, ESIM_dist

        elif similarity_model_name =='InferSent':
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'InferSent_dist.npy'))
            ESIM_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'ESIM_dist.npy'))
            return encoder_models.InferSentEncoder(), None, load_ESIM_model(ESIM_model_name), \
                    None, None, \
                    similarity_dist, None, ESIM_dist

    elif reward_function == 'PARASIM_F':
        if similarity_model_name =='BERT':
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'BERT_dist.npy'))
            fluency_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'fluency_dist.npy'))
            ESIM_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'ESIM_dist.npy'))
            return encoder_models.BERTEncoder(), encoder_models.GPTLanguageModel(),
load_ESIM_model(ESIM_model_name), \
                    None, None, \
                    similarity_dist, fluency_dist, ESIM_dist

        elif similarity_model_name =='InferSent':
            similarity_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'InferSent_dist.npy'))
            fluency_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path,
'fluency_dist.npy'))
            ESIM_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'ESIM_dist.npy'))
            return encoder_models.InferSentEncoder(), encoder_models.GPTLanguageModel(),
load_ESIM_model(ESIM_model_name), \
                    None, None, \
                    similarity_dist, fluency_dist, ESIM_dist

    elif reward_function == 'ESIM':
        return None, None, load_ESIM_model(ESIM_model_name), \
                None, None, \
                None, None, None

    elif reward_function == 'ESIM_short':
        return None, None, load_ESIM_model(ESIM_model_name), \
                None, None, \
                None, None, None

    elif reward_function == 'ESIM_rare':
        return None, None, load_ESIM_model(ESIM_model_name), \
                None, None, \
                None, None, None

    elif reward_function == 'ESIM_F':
        fluency_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'fluency_dist.npy'))
        ESIM_dist = data.load_np_data(os.path.join(config.saved_SM_dist_path, 'ESIM_dist.npy'))
```

```
        return None, encoder_models.GPTLanguageModel(), load_ESIM_model(ESIM_model_name), \
                None, None, \
                None, fluency_dist, ESIM_dist

    elif reward_function == 'ESIM_syllables':
        return None, None, load_ESIM_model(ESIM_model_name), \
                None, None, \
                None, None, None

    elif reward_function == 'ESIM_libertarian':
        logr_model = data.load_vocab_index(os.path.join(
                config.saved_reddit_model_path, 'anarsoc_libertar_logr.pickle'))

        std_scaler = data.load_vocab_index(os.path.join(
                config.saved_reddit_model_path, 'std_scaler.pickle'))

        return encoder_models.BERTEncoder(), None, load_ESIM_model(ESIM_model_name), \
                logr_model, std_scaler, \
                None, None, None

    else:
        return None, None, None, None, None, None, None, None

#%% ------------------------------- ARCHIVE -------------------------------

# Prior defined versions of BLEU and ROUGE

#from collections import Counter
#from rouge.rouge import rouge_n_sentence_level
#from nltk import bigrams
#
#def BLEU_score(target_tokens, pred_tokens, ngram = 'unigram'):
#    if ngram == 'bigram':
#        target_tokens = list(bigrams(target_tokens))
#        pred_tokens = list(bigrams(pred_tokens))
#
#    word_counts = Counter(target_tokens)
#    score = 0
#
#    for token in pred_tokens:
#        if word_counts[token] > 0:
#           word_counts[token] -=1
#           score += 1
#    score /= len(target_tokens)
#    return score
#
#def ROUGE_score(target, pred, ngram = 'unigram'):
#    """ Note: rouge_n_sentence_level has hypothesis and reference positions swapped"""
#    if ngram == 'unigram':
#        _, _, score = rouge_n_sentence_level(pred, target, 1)
#        return score
#    if ngram == 'bigram':
#        _, _, score = rouge_n_sentence_level(pred, target, 2)
#        return score
#    else:
#        print("Please select either: 'unigram' or 'bigram'")
```

9.5 RL Model

```
"""Defines and trains RL models for ParaPhrasee environment"""

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
from torch.distributions import Categorical

import numpy as np
import os
import argparse
import logging

import config
import data
import utils
```

```
import supervised_model as sm
from train_ESIM import RLAdversary, load_ESIM_model
import paraphrasee_env

import MCTS

DEVICE = config.DEVICE

MAX_LENGTH = config.MAX_LENGTH

SOS_token = config.SOS_token
EOS_token = config.EOS_token

#Load data
train_pairs = data.TRAIN_PAIRS
val_pairs = data.VAL_PAIRS
test_pairs = data.TEST_PAIRS
vocab_index = data.VOCAB_INDEX

# Define command line arguments for experiment
parser = argparse.ArgumentParser(description='Train_ParaPhrasee_Model')
parser.add_argument('--train_models', action='store_true', help='enable training of RL models')
parser.add_argument('--test_MCTS', action='store_true', help='enable testing of RL models')
parser.add_argument('--folder_name', type=str,
                    help='Brief description of experiment (no spaces)')
parser.add_argument('--checkpoint_n_episodes', type=int, default=5000,
                    help='Brief description of experiment (no spaces)')
parser.add_argument('--reward_function', type=str, default='BLEU1',
                    choices=config.perf_metrics_list,
                    help='select reward function')
parser.add_argument('--similarity_model_name', type=str, default='BERT',
                    choices=['BERT', 'InferSent'],
                    help='select reward function')

parser.add_argument('--use_pretrained_critic', type=int, choices={0, 1}, default=1,
                    help='critic is initialized with pretrained model')
parser.add_argument('--pretrain_critic_n_episodes', type=int, default=0,
                    help='number of iterations to pretrain the critic (default: 0)')
parser.add_argument('--n_episodes', type=int, default=30000,
                    help='max number of iterations to train the RL model (default: 2500)')
parser.add_argument('--verbose', action='store_true', help='print results during training')

parser.add_argument('--init_critic', type=int, choices={0, 1}, default=1, help='initializes critic
model')
parser.add_argument('--transfer_weights', type=int, choices={0, 1}, default=1,
                    help='transfers weights from supervised model to actor')
parser.add_argument('--use_policy_distillation', type=int, choices={0, 1}, default=0,
                    help='adds policy distillation error to reward function')
parser.add_argument('--MCTS_thresh', type=float, default=0,
                    help='Uses MCTS unless max certainty is above specified prob (default: 0)')
parser.add_argument('--use_adversarial_training', type=int, choices={0, 1}, default=0,
                    help='update adversary')

# Mostly for helper functions and debugging
parser.add_argument('--update_RL_models', type=int, choices={0, 1}, default=1,
                    help='allows the update of the rl models')
parser.add_argument('--use_MLE', type=int, choices={0, 1}, default=0,
                    help='can use MLE instead of sample')

parser.add_argument('--load_models', action='store_true', help='Load pretrained model from prior
point')
parser.add_argument('--load_model_folder_name', type=str,
                    help='folder which contains the saved models to be used')

args = parser.parse_args()
args.env_name = 'ParaPhrasee'
args.save_models = 0

if args.train_models:
    args.save_models = 1
    saved_RL_model_results = data.SaveRLModelResults(args.env_name, args.folder_name)
    saved_RL_model_results.check_folder_exists()

args.SM_FOLDER = 'VanillaEncoder'
args.SM_ENCODER_FILE_NAME = 'encoder_3.150.pt'
args.SM_DECODER_FILE_NAME = 'decoder_3.150.pt'
```

```
args.PRETRAINED_CRITIC = args.reward_function+'_pretrained_critic_125k.pt'

def set_early_stopping_thresh(reward_function):
    if reward_function in config.perf_metrics_list:
        return 10.00
    else:
        print("Please specify one of the following reward functions: {}".format(
                config.perf_metrics_list))

args.early_stopping_reward_thresh = set_early_stopping_thresh(args.reward_function)

#%%
#args.train_models = 1
#args.verbose = 1
#saved_RL_model_results = data.SaveRLModelResults('ParaPhrasee', 'Test')
#args.reward_function = 'BLEU1'

#args.update_RL_models = 0
#args.MCTS_thresh = 0.90

#%%

class RLActor(nn.Module):
    """Vanilla decoder which decodes based on single context vector"""
    def __init__(self, embedding_size, hidden_size, output_size):
        super(RLActor, self).__init__()
        self.name = 'VanillaDecoderRNNActor'
        self.is_agent = True
        self.uses_attention = False
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, self.embedding_size)
        self.gru = nn.GRU(self.embedding_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

        self.gamma = 0.9999
        self.saved_action_values = []
        self.rewards = []

    def forward(self, input, hidden, temperature=1):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]) / temperature)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class RLCritic(nn.Module):
    """Critic which predicts the value of a given state"""
    def __init__(self, embedding_size, hidden_size, output_size):
        super(RLCritic, self).__init__()
        self.name = 'CriticRNN'
        self.is_agent = True
        self.uses_attention = False
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, self.embedding_size)
        self.gru = nn.GRU(self.embedding_size, hidden_size)
        self.out = nn.Linear(hidden_size, 1)

        self.saved_state_values = []

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.out(output[0])
        return output

    def initHidden(self):
```

```
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class TeacherRNN(nn.Module):
    """Vanilla decoder which decodes based on single context vector,
        Has the same architecture as the DecoderRNN - the only difference is the addition
        of temperature and softmax instead of log softmax"""
    def __init__(self, embedding_size, hidden_size, output_size):
        super(TeacherRNN, self).__init__()
        self.name = 'VanillaDecoderRNN'
        self.is_agent = False
        self.uses_attention = False
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, self.embedding_size)
        self.gru = nn.GRU(self.embedding_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, input, hidden, temperature=1):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]) / temperature)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

def select_action(input_state, input_hidden_state, actor_model, critic_model=None,
                  teacher_model=None, K=1, use_MLE=False, MCTS_thresh=0):
    """Applies the model on a given input and hidden state to make a prediction of which action to take
        Can use MLE, MCTS, or sampling to select an action"""
    probs, hidden_state = actor_model(input_state, input_hidden_state)
    m = Categorical(probs)

    # Use MLE instead of sampling distribution
    if use_MLE:
        _, topi = probs.data.topk(1)
        action = topi.squeeze()

    # Note: MCTS only works during validation (when the model is not tracking gradients)
    elif torch.max(probs).detach() < MCTS_thresh:
        action, hidden_state, _ = MCTS.UCT_search(
                env, input_state, input_hidden_state, actor_model, critic_model,
                5, env.action_space, 100)
        action = torch.tensor(action, device=config.DEVICE)

    else:
        action = m.sample()

    actor_model.saved_action_values.append(m.log_prob(action))

    if critic_model != None:
        state_value = critic_model(input_state, input_hidden_state)
        critic_model.saved_state_values.append(state_value)

    if teacher_model != None:
        # Add policy distillation error
        actor_probs, _ = actor_model(input_state, input_hidden_state, K)
        supervised_probs, _ = teacher_model(input_state, input_hidden_state, K)
        KL_error = utils.KL_divergence(actor_probs, supervised_probs, K)
        return action, hidden_state, KL_error.item()

    return action, hidden_state, None


#%%

def REINFORCE_update(actor_model, actor_optimizer):
    """Update the model when using REINFORCE instead of Actor-Critic"""
    R = 0
    policy_loss = []
    returns = []

    # Discount the rewards back to present
```

```
    for r in actor_model.rewards[::-1]:
        R = r + actor_model.gamma * R
        returns.insert(0, R)

    # Scale the rewards
    returns = torch.tensor(returns)
    ###returns = (returns - returns.mean()) / (returns.std() + EPS)

    # Calculate the loss
    for log_prob, R in zip(actor_model.saved_action_values, returns):
        policy_loss.append(-log_prob * R)

    # Update network weights
    actor_optimizer.zero_grad()
    policy_loss = torch.cat(policy_loss).sum()
    policy_loss.backward()
    actor_optimizer.step()

    # Clear memory
    del actor_model.rewards[:]
    del actor_model.saved_action_values[:]

def actor_critic_update(actor_model, actor_optimizer, critic_model, critic_optimizer,
                        only_update_critic=False):
    """Update the model when using Actor-Critic"""
    R = 0
    saved_actions = actor_model.saved_action_values
    saved_states = critic_model.saved_state_values
    policy_losses = [] # list to save actor (policy) loss
    value_losses = [] # list to save critic (value) loss
    returns = [] # list to save the true values

    # calculate the true value using rewards returned from the environment
    for r in actor_model.rewards[::-1]:
        # calculate the discounted value
        R = r + actor_model.gamma * R
        returns.insert(0, R)

    # Scale the rewards
    returns = torch.tensor(returns)
    ###returns = (returns - returns.mean()) / (returns.std() + EPS) # scaling reduced performance

    for log_prob, value, R in zip(saved_actions, saved_states, returns):
        advantage = R - value.item()

        # calculate actor (policy) loss
        policy_losses.append(-log_prob * advantage)

        # calculate critic (value) loss using L1 smooth loss
        value_losses.append(F.smooth_l1_loss(value, torch.tensor([R], device=DEVICE)))

    # reset gradients
    actor_optimizer.zero_grad()
    critic_optimizer.zero_grad()

    if only_update_critic:
        loss = torch.stack(value_losses).sum()

        # perform backprop
        loss.backward()
        critic_optimizer.step()

    else:
        # sum up all the values of policy_losses and value_losses
        loss = torch.stack(policy_losses).sum() + torch.stack(value_losses).sum()

        # perform backprop
        loss.backward()
        actor_optimizer.step()
        critic_optimizer.step()

    # reset rewards and action buffer
    del actor_model.rewards[:]
    del actor_model.saved_action_values[:]
    del critic_model.saved_state_values[:]
```

```
#%%

def init_actor_critic_models(supervised_decoder, init_critic=True, transfer_weights=True):
    """Instantiates the actor and critic models as well as the optimizers"""
    # Define actor and critic
    actor_model = RLActor(supervised_decoder.embedding_size, supervised_decoder.hidden_size,
                           vocab_index.n_words).to(DEVICE)

    # Transfer weights to actor and set optimizer
    if transfer_weights:
        actor_model.load_state_dict(supervised_decoder.state_dict())

    actor_optimizer = optim.SGD(actor_model.parameters(), lr=0.001)

    if init_critic:
        critic_model = RLCritic(supervised_decoder.embedding_size, supervised_decoder.hidden_size,
                                vocab_index.n_words).to(DEVICE)
        critic_optimizer = optim.SGD(critic_model.parameters(), lr=0.001)

    else:
        critic_model = None
        critic_optimizer = None

    return actor_model, critic_model, actor_optimizer, critic_optimizer

def train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                    supervised_encoder, teacher_model,
                    use_policy_distillation, update_RL_models, only_update_critic,
                    use_MLE, MCTS_thresh, n_episodes):
    """Main training loop to train the actor and critic models"""

    for i_episode in range(1, n_episodes+1):
        (prev_action, hidden_state), ep_reward, done = env.reset(), 0, False

        ep_env_reward = 0
        ep_KL_penalty = 0

        for step_i in range(1, env.max_steps+1):
            action, hidden_state, KL_error = select_action(input_state=prev_action,
input_hidden_state=hidden_state,
                                                           actor_model=actor_model,
critic_model=critic_model,
                                                           teacher_model=teacher_model, K=hp.K,
use_MLE=use_MLE,
                                                           MCTS_thresh=MCTS_thresh)

            (prev_action, hidden_state), env_reward, done, _ = env.step(action, hidden_state)

            if use_policy_distillation:
                avg_RL_reward = np.mean(saved_RL_model_results.env_rewards[-hp.distillation_n_mean:]) \
                        if len(saved_RL_model_results.env_rewards) > hp.distillation_n_mean else 0

                lambda_value= utils.lambda_value(beta=hp.beta,
sm_baseline_reward=hp.sm_baseline_reward,
                                             avg_rewards=avg_RL_reward)
                KL_penalty = lambda_value * -KL_error
                reward = env_reward + KL_penalty
                ep_env_reward += env_reward
                ep_KL_penalty += KL_penalty

            else:
                reward = env_reward
                ep_reward += reward

            actor_model.rewards.append(reward)

            if done:
                if args.use_adversarial_training:
                    adversary_model.pred_pairs.append([env.source_sentence, env.pred_sentence()])
                break

        if update_RL_models:
            if critic_model != None:
                actor_critic_update(actor_model, actor_optimizer, critic_model, critic_optimizer,
                                only_update_critic=only_update_critic)
            else:
```

```
                REINFORCE_update(actor_model, actor_optimizer)

        if use_policy_distillation:
            saved_RL_model_results.env_rewards.append(ep_env_reward)
            saved_RL_model_results.KL_penalty.append(ep_KL_penalty)

            if args.verbose and (i_episode % hp.print_every == 0):
                avg_env_reward = np.mean(saved_RL_model_results.env_rewards[-hp.print_every:])
                avg_KL_penalty = np.mean(saved_RL_model_results.KL_penalty[-hp.print_every:])

                print('Episode {} | Avg env reward: {:.2f} | Avg KL penalty: {:.2f} | Lambda value:
{:.2f}'.format(
                      i_episode, avg_env_reward, avg_KL_penalty, lambda_value))

            early_stopping_value = np.mean(saved_RL_model_results.env_rewards[-
hp.early_stopping_n_mean:]) \
                    if len(saved_RL_model_results.env_rewards) > hp.early_stopping_n_mean else 0
            if (early_stopping_value >= hp.early_stopping_reward_thresh) or (i_episode == n_episodes-
1):
                if args.save_models:
                    saved_RL_model_results.save_top_models(actor_model,
'actor_{:.3f}.pt'.format(early_stopping_value))
                    if args.init_critic:
                        saved_RL_model_results.save_top_models(critic_model,
'critic_{:.3f}.pt'.format(early_stopping_value))
                    saved_RL_model_results.export_rewards('model_performance.txt')

                    if args.use_adversarial_training:
                        model_name = 'adversary_model{}_{:.3}.pt'.format(
                            adversary_model.update_iter,
                            adversary_model.training_accuracy[adversary_model.update_iter][-1])
                        data.save_model(adversary_model.model,
                                        os.path.join(saved_RL_model_results.folder_path, model_name))
                break

        else:
            saved_RL_model_results.env_rewards.append(ep_reward)

            if args.verbose and (i_episode % hp.print_every == 0):
                avg_env_reward = np.mean(saved_RL_model_results.env_rewards[-hp.print_every:])
                print('Episode {} | Average reward: {:.2f}'.format(i_episode, avg_env_reward))

            early_stopping_value = np.mean(saved_RL_model_results.env_rewards[-
hp.early_stopping_n_mean:]) \
                    if len(saved_RL_model_results.env_rewards) > hp.early_stopping_n_mean else 0

            if args.save_models and (i_episode % args.checkpoint_n_episodes == 0):
                saved_RL_model_results.save_top_models(actor_model, 'actor_iter{}_{:.3f}.pt'.format(
                        i_episode, early_stopping_value))
                if args.init_critic:
                    saved_RL_model_results.save_top_models(
                            critic_model, 'critic_iter{}_{:.3f}.pt'.format(i_episode,
early_stopping_value))

            if (early_stopping_value >= hp.early_stopping_reward_thresh) or (i_episode == n_episodes-
1):
                if args.save_models:
                    saved_RL_model_results.save_top_models(actor_model,
'actor_{:.3f}.pt'.format(early_stopping_value))
                    if args.init_critic:
                        saved_RL_model_results.save_top_models(critic_model,
'critic_{:.3f}.pt'.format(early_stopping_value))
                    saved_RL_model_results.export_rewards('model_performance.txt')

                    if args.use_adversarial_training:
                        model_name = 'adversary_model{}_{:.3}.pt'.format(
                            adversary_model.update_iter,
                            adversary_model.training_accuracy[adversary_model.update_iter][-1])
                        data.save_model(adversary_model.model,
                                        os.path.join(saved_RL_model_results.folder_path, model_name))
                break

        if (i_episode % hp.update_adversary_every == 0) and args.use_adversarial_training:
            n_target_samples = len(adversary_model.pred_pairs) / 0.7 - len(adversary_model.pred_pairs)
            adversary_model.target_pairs = data.sample_list(env.sentence_pairs,
n_samples=int(n_target_samples))
```

```
            adversary_model.update_model()

            env.ESIM_model = adversary_model.model

class HyperParams(object):
    """Sets the experiment hyperparameters"""
    def __init__(self, print_every=10, early_stopping_reward_thresh=0.50):
        self.print_every = print_every
        self.K = 5
        self.early_stopping_reward_thresh = early_stopping_reward_thresh
        self.early_stopping_n_mean = 50

        self.beta = 10000
        self.sm_baseline_reward = 0.25
        self.distillation_n_mean = 50

        self.update_adversary_every = 6000

#%%

if args.train_models:
    """Instantiates models and environment, trains and evaluates the model"""

    # Instantiate models and environment
    supervised_encoder, supervised_decoder = sm.load_supervised_models(
            args.SM_FOLDER, encoder_file_name=args.SM_ENCODER_FILE_NAME,
decoder_file_name=args.SM_DECODER_FILE_NAME)
    actor_model, critic_model, actor_optimizer, critic_optimizer = init_actor_critic_models(
            supervised_decoder, init_critic=args.init_critic, transfer_weights=args.transfer_weights)

    # Create folder if saving models
    if args.save_models:
        saved_RL_model_results.init_folder(args, actor_model, critic_model)

    # Load pretrained critic
    if args.use_pretrained_critic and critic_model is not None:
        data.load_model(critic_model, os.path.join(config.saved_RL_model_path, args.env_name,
                                                   args.reward_function, args.PRETRAINED_CRITIC))
    # Optionally load trained models
    if args.load_models:
        actor_model, critic_model = data.load_RL_models(
                args.env_name, args.load_model_folder_name, actor_model, critic_model,
                actor_file_name='best', critic_file_name='best')

    # Instantiate teacher model if using policy distillation
    if args.use_policy_distillation:
        teacher_model = TeacherRNN(embedding_size=supervised_decoder.embedding_size,
                                   hidden_size=supervised_decoder.hidden_size,
                                   output_size=vocab_index.n_words).to(DEVICE)
        teacher_model.load_state_dict(supervised_decoder.state_dict())
    else:
        teacher_model = None

    # Load adversarial model if use adversarial training
    if args.use_adversarial_training:
        adversary_model = RLAdversary('ESIM_noisy_3')

    # Instantiate the environment and hyperparameters
    input_sentence = train_pairs[0]
    env = paraphrasee_env.ParaPhraseeEnvironment(
            source_sentence=input_sentence[0], target_sentence=input_sentence[1],
            supervised_encoder=supervised_encoder, reward_function=args.reward_function,
            similarity_model_name=args.similarity_model_name, sentence_pairs=train_pairs)
    hp = HyperParams(print_every=10, early_stopping_reward_thresh=args.early_stopping_reward_thresh)

    # Train models and save checkpoint if error occurs
    try:
        if args.pretrain_critic_n_episodes > 0:
            train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                            supervised_encoder, teacher_model,
                            use_policy_distillation=False, update_RL_models=True,
                            only_update_critic=True, use_MLE=False, MCTS_thresh=0,
                            n_episodes=args.pretrain_critic_n_episodes)
        else:
            train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                                supervised_encoder, teacher_model,
```

```
                                use_policy_distillation=args.use_policy_distillation,
                                update_RL_models=args.update_RL_models,
                                only_update_critic=False, use_MLE=args.use_MLE,
                                MCTS_thresh=args.MCTS_thresh, n_episodes=args.n_episodes)
    except:
        if args.save_models:
            saved_RL_model_results.save_top_models(actor_model, 'actor_CHECKPOINT.pt')
            if args.init_critic:
                saved_RL_model_results.save_top_models(critic_model, 'critic_CHECKPOINT.pt')
            saved_RL_model_results.export_rewards('model_performance.txt')

            if args.use_adversarial_training:
                model_name = 'adversary_model{}_{:.3}.pt'.format(
                    adversary_model.update_iter,
                    adversary_model.training_accuracy[adversary_model.update_iter][-1])
                data.save_model(adversary_model.model,
                                os.path.join(saved_RL_model_results.folder_path, model_name))

        logging.error("Exception occurred", exc_info=True)

#%%

def validation_perf(input_folder_name, val_pairs, n_episodes, reward_metric='BLEU1',
similarity_model_name='BERT',
                    use_MLE=True, MCTS_thresh=0, set_ESIM_model=None, verbose=False):
    """Instantiates and loads trained models and environment in order to test the model performance"""

    supervised_encoder, supervised_decoder = sm.load_supervised_models(
            args.SM_FOLDER, encoder_file_name=args.SM_ENCODER_FILE_NAME,
decoder_file_name=args.SM_DECODER_FILE_NAME)

    actor_model, critic_model, _, _ = init_actor_critic_models(
            supervised_decoder, init_critic=1, transfer_weights=0)

    actor_model, critic_model = data.load_RL_models(
                args.env_name, input_folder_name, actor_model, critic_model,
                actor_file_name='best', critic_file_name='best')

    teacher_model = None

    input_sentence = train_pairs[0]
    env = paraphrasee_env.ParaPhraseeEnvironment(
            source_sentence=input_sentence[0], target_sentence=input_sentence[1],
            supervised_encoder=supervised_encoder, reward_function=reward_metric,
            similarity_model_name=similarity_model_name, sentence_pairs=val_pairs)
    if set_ESIM_model is not None:
        env.ESIM_model = set_ESIM_model

    hp = HyperParams(print_every=10, early_stopping_reward_thresh=args.early_stopping_reward_thresh)

    validation_performance = []
    validation_sentences = []

    with torch.no_grad():
        for i_episode in range(1, n_episodes+1):
            (prev_action, hidden_state), ep_reward, done = env.reset(), 0, False

            for step_i in range(1, env.max_steps+1):
                action, hidden_state, KL_error = select_action(input_state=prev_action,
input_hidden_state=hidden_state,
                                                               actor_model=actor_model,
critic_model=critic_model,
                                                               teacher_model=teacher_model, K=hp.K,
use_MLE=use_MLE,
                                                               MCTS_thresh=MCTS_thresh)

                (prev_action, hidden_state), env_reward, done, _ = env.step(action, hidden_state)

                reward = env_reward
                ep_reward += reward

                actor_model.rewards.append(reward)

                if done:
                    break
```

```
            validation_performance.append(ep_reward)
            validation_sentences.append([env.source_sentence, env.target_sentence,
env.pred_sentence()])

            if verbose:
                print('Source sentence: ', env.source_sentence)
                print('Target sentence: ', env.target_sentence)
                print()
                print('Supervised model prediction: ', env.supervised_baseline(supervised_decoder))
                print('RL model prediction: ', env.pred_sentence(), ep_reward)
                print()

    return np.array(validation_sentences), np.mean(validation_performance), validation_performance

if args.test_MCTS:
    """Designed as one-off to evaluate performance of MCTS"""
    input_folder_name=args.folder_name
    val_pairs=test_pairs
    n_episodes=args.n_episodes
    reward_metric='ESIM'
    similarity_model_name='BERT'
    use_MLE=False
    MCTS_thresh=0.80
    set_ESIM_model=load_ESIM_model(folder_name='ESIM_adv_30k1', file_name='ESIM_0.755.pt')
    verbose=False

    supervised_encoder, supervised_decoder = sm.load_supervised_models(
            args.SM_FOLDER, encoder_file_name=args.SM_ENCODER_FILE_NAME,
decoder_file_name=args.SM_DECODER_FILE_NAME)

    actor_model, critic_model, _, _ = init_actor_critic_models(
            supervised_decoder, init_critic=1, transfer_weights=0)

    actor_model, critic_model = data.load_RL_models(
                args.env_name, input_folder_name, actor_model, critic_model,
                actor_file_name='actor_0.391.pt', critic_file_name='critic_0.391.pt')

    teacher_model = None

    input_sentence = train_pairs[0]
    env = paraphrasee_env.ParaPhraseeEnvironment(
            source_sentence=input_sentence[0], target_sentence=input_sentence[1],
            supervised_encoder=supervised_encoder, reward_function=reward_metric,
            similarity_model_name=similarity_model_name, sentence_pairs=val_pairs)
    if set_ESIM_model is not None:
        env.ESIM_model = set_ESIM_model

    hp = HyperParams(print_every=10, early_stopping_reward_thresh=args.early_stopping_reward_thresh)

    validation_performance = []
    validation_sentences = []

    try:
        with torch.no_grad():
            for i_episode in range(1, n_episodes+1):
                (prev_action, hidden_state), ep_reward, done = env.reset(), 0, False

                for step_i in range(1, env.max_steps+1):
                    action, hidden_state, KL_error = select_action(input_state=prev_action,
input_hidden_state=hidden_state,
                                                                   actor_model=actor_model,
critic_model=critic_model,
                                                                   teacher_model=teacher_model, K=hp.K,
use_MLE=use_MLE,
                                                                   MCTS_thresh=MCTS_thresh)

                    (prev_action, hidden_state), env_reward, done, _ = env.step(action, hidden_state)

                    reward = env_reward
                    ep_reward += reward

                    actor_model.rewards.append(reward)

                    if done:
                        break
```

```
                validation_performance.append(ep_reward)
                validation_sentences.append([env.source_sentence, env.target_sentence,
env.pred_sentence()])

                if verbose:
                    print('Source sentence: ', env.source_sentence)
                    print('Target sentence: ', env.target_sentence)
                    print()
                    print('Supervised model prediction: ', env.supervised_baseline(supervised_decoder))
                    print('RL model prediction: ', env.pred_sentence(), ep_reward)
                    print()

        data.save_np_data(validation_sentences, os.path.join(
                config.saved_RL_text_path, 'MCTS/', reward_metric+'_MCTS.npy'))

    except:
        print(validation_sentences)
        data.save_np_data(validation_sentences, os.path.join(
                config.saved_RL_text_path, 'MCTS/', reward_metric+'_MCTS.npy'))
```

9.6 Toy RL Pipeline

```
"""Defines and trains RL models for either CartPole or FrozenLake environments"""

import numpy as np
import time

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical

from collections import deque
import os
import argparse
from copy import deepcopy

import config
import data
import cart_pole_env
import frozen_lake_env
import utils
import MCTS

DEVICE = config.DEVICE
EPS = config.EPS
torch.manual_seed(config.SEED)

# Define command line arguments for experiment
parser = argparse.ArgumentParser(description='Train_RL_Model')
parser.add_argument('--train_models', action='store_true', help='enable training of RL models')
parser.add_argument('--folder_name', type=str,
                    help='Brief description of experiment (no spaces)')

parser.add_argument('--sparse_env', type=int, choices={0, 1}, default=1,
                    help='sparsifies environment rewards')
parser.add_argument('--relative_rewards', type=int, choices={0, 1}, default=0,
                    help='use rewards relative to supervised model')

parser.add_argument('--env_name', type=str, default='CartPole',
                    choices=['CartPole', 'FrozenLake'], help='select RL environment')
parser.add_argument('--use_pretrained_critic', type=int, choices={0, 1}, default=1,
                    help='critic is initialized with pretrained model')
parser.add_argument('--pretrain_critic_n_episodes', type=int, default=0,
                    help='number of iterations to pretrain the critic (default: 0)')
parser.add_argument('--n_episodes', type=int, default=2500,
                    help='max number of iterations to train the RL model (default: 2500)')
parser.add_argument('--verbose_training', type=int, choices={0, 1}, default=0,
                    help='print results during training')

parser.add_argument('--init_critic', type=int, choices={0, 1}, default=1, help='initializes critic
model')
```

```
parser.add_argument('--transfer_weights', type=int, choices={0, 1}, default=1,
                    help='transfers weights from supervised model to actor')
parser.add_argument('--use_policy_distillation', type=int, choices={0, 1}, default=0,
                    help='adds policy distillation error to reward function')
parser.add_argument('--MCTS_thresh', type=float, default=0,
                    help='Uses MCTS unless max certainty is above specified prob (default: 0)')

# Mostly for helper functions and debugging
parser.add_argument('--update_RL_models', type=int, choices={0, 1}, default=1,
                    help='allows the update of the rl models')
parser.add_argument('--use_MLE', type=int, choices={0, 1}, default=0,
                    help='can use MLE instead of sample')

parser.add_argument('--load_models', action='store_true', help='Load pretrained model from prior
point')
parser.add_argument('--load_model_folder_name', type=str,
                    help='folder which contains the saved models to be used')

args = parser.parse_args()
args.save_models = 0

if args.train_models:
    args.save_models = 1
    saved_RL_model_results = data.SaveRLModelResults(args.env_name, args.folder_name)
    saved_RL_model_results.check_folder_exists()

# Specify which encoder / decoder is used
args.FROZENLAKE_ENCODER = 'FrozenLake/FrozenLakeEncoder_medium.pt'
args.FROZENLAKE_DECODER = 'FrozenLake/FrozenLakeDecoder_medium.pt'

args.CARTPOLE_DECODER = 'CartPole/CartpoleDecoder.pt'
args.CP_PRETRAINED_CRITIC = 'CP_critic_model_585.pt'
args.FL_PRETRAINED_CRITIC = 'FL_critic_model_5000.pt'

#%% Manual Testing - turn train models on while keeping save models off then modify as you like

#Run test
#args.train_models = 1
#args.env_name = 'FrozenLake'
#args.verbose_training = 1
#saved_RL_model_results = data.SaveRLModelResults(args.env_name, 'Test')
#
#args.n_episodes = 3000
#args.load_models = 1
#args.load_model_folder_name = os.path.join(config.saved_RL_model_path,
'FrozenLake/Medium/Test3/Transfer_Weights10/')
#args.update_RL_models = 0
#args.use_MLE = 0
#args.MCTS_thresh = 0.65


#%% Load environment data

def load_FrozenLake_data():
    """Loads FrozenLake training data"""
    input_map_df = data.load_np_data('Data/RL_Data/FL_input_map_df.npy')
    visited_states_df = np.load('Data/RL_Data/FL_states_df.npy', allow_pickle=True)
    selected_actions_df = np.load('Data/RL_Data/FL_selected_action_sequence_df.npy', allow_pickle=True)

    train_data, val_data, test_data = data.train_test_split(
            list(zip(input_map_df, visited_states_df, selected_actions_df)))

    assert all([len(train_data[i][1]) == len(train_data[i][2]) for i in range(len(train_data))]), \
        'Lengths of states and actions are not equal'

    return train_data, val_data, test_data

def load_CartPole_data():
    """Loads CartPole training data"""
    visited_states_df = np.load('Data/RL_Data/CP_states_df.npy')
    selected_actions_df = np.load('Data/RL_Data/CP_selected_action_sequence_df.npy')
    input_map_df = np.zeros((len(visited_states_df),1))

    train_data, val_data, test_data = data.train_test_split(
            list(zip(input_map_df, visited_states_df, selected_actions_df)))
```

```
    assert all([len(train_data[i][1]) == len(train_data[i][2]) for i in range(len(train_data))]), \
        'Lengths of states and actions are not equal'

    return train_data, val_data, test_data

def load_env_data(env_name):
    """Initializes data for relevant environment"""
    if env_name == 'CartPole':
        train_data, val_data, test_data = load_CartPole_data()

    elif env_name == 'FrozenLake':
        train_data, val_data, test_data = load_FrozenLake_data()

    else:
        print("Please select one of the following environments: ['FrozenLake', 'CartPole']")

    return train_data, val_data, test_data

train_data, val_data, test_data = load_env_data(args.env_name)

#%% Define supervised models

class MLPStateEncoder(nn.Module):
    """Uses a MLP state encoder for FrozenLake"""
    def __init__(self, input_size, hidden_size, output_size):
        super(MLPStateEncoder, self).__init__()
        self.name = 'MLPStateEncoder'
        self.hidden_size = hidden_size

        self.h1 = nn.Linear(input_size, self.hidden_size)
        self.dropout = nn.Dropout(p=0.6)
        self.h2 = nn.Linear(self.hidden_size, output_size)

    def forward(self, x):
        x = torch.tensor(x, dtype=torch.float32, device=DEVICE).view(1,1,-1)
        x = self.h1(x)
        x = self.dropout(x)
        x = F.relu(x)
        action_scores = self.h2(x)
        return action_scores

class CNNStateEncoder(nn.Module):
    """CNN encoder model designed for FrozenLake with a 5x5 input state space.
        Would need to manually update hyperparameters for a different state space size"""
    def __init__(self, output_size):
        super(CNNStateEncoder, self).__init__()
        self.name = 'CNNStateEncoder'
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1)
        self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 16, kernel_size=3, stride=2, padding=1)
        self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.linear_output = nn.Linear(144, output_size)

    def forward(self, x):
        x = torch.tensor(x, dtype=torch.float32, device=DEVICE).view(1,1,5,5)
        x = F.relu(self.conv1(x))
        x = self.maxpool1(x)
        x = F.relu(self.conv2(x))
        x = x.view(1,1,-1)
        output = self.linear_output(x)
        return output

class GeneralDecoderRNN(nn.Module):
    """Vanilla decoder (WITH NO EMBEDDINGS) which decodes based on single context vector"""
    def __init__(self, input_size, hidden_size, output_size):
        super(GeneralDecoderRNN, self).__init__()
        self.name = 'GeneralDecoderRNN'
        self.hidden_size = hidden_size

        self.gru = nn.GRU(input_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = torch.tensor(input, dtype=torch.float32, device=DEVICE).view(1,1,-1)
        output, hidden = self.gru(output, hidden)
```

```
        output = self.softmax(self.out(output[0]))
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class TeacherRNN(nn.Module):
    """Vanilla decoder (WITH NO EMBEDDINGS) which decodes based on single context vector
        Has the same architecture as the General Decoder RNN - the only difference is the addition
        of temperature and softmax instead of log softmax"""
    def __init__(self, input_size, hidden_size, output_size):
        super(TeacherRNN, self).__init__()
        self.name = 'GeneralDecoderRNN'
        self.hidden_size = hidden_size

        self.gru = nn.GRU(input_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, input, hidden, temperature=1):
        output = torch.tensor(input, dtype=torch.float32, device=DEVICE).view(1,1,-1)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]) / temperature)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

#%% Train and evaluate models

def train_supervised(input_map, visited_states, selected_actions, encoder, decoder,
          encoder_optimizer, decoder_optimizer, criterion):
    """Train supervised models for CartPole or FrozenLake"""
    if encoder != None:
        encoder.zero_grad()
        hidden_state = encoder(input_map)

    else:
        hidden_state = decoder.initHidden()

    decoder.zero_grad()

    target_length = len(selected_actions)
    loss = 0

    for i in range(target_length):
        state_input = visited_states[i]
        model_output, hidden_state = decoder(
            state_input, hidden_state)

        loss += criterion(model_output, torch.tensor([selected_actions[i]],
                                                     dtype=torch.long, device=DEVICE))

    loss.backward()
    if encoder != None:
        encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length

def trainIters(input_data, encoder, decoder,
               encoder_optimizer, decoder_optimizer, n_iters=20, print_every=10):
    """Applies training loop to train models on data"""
    if encoder != None:
        encoder.train()
    decoder.train()
    start = time.time()

    print_loss_total = 0  # Reset every print_every
    criterion = nn.NLLLoss()

    # Sample n random pairs
    selected_indices = np.random.choice(len(input_data), n_iters, replace=False)

    # For EACH pair train model to decrease loss
    for idx, selected_idx in enumerate(selected_indices):
```

```
        loss = train_supervised(input_data[selected_idx][0], input_data[selected_idx][1],
                     input_data[selected_idx][2], encoder, decoder,
                     encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss

        iter = idx+1
        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (utils.timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100, print_loss_avg))

def evaluate(input_map, visited_states, selected_actions, encoder, decoder, criterion):
    """Evaluate the performance of the trained models"""
    with torch.no_grad():
        if encoder != None:
            hidden_state = encoder(input_map)
        else:
            hidden_state = decoder.initHidden()

        target_length = len(visited_states)
        loss = 0

        for i in range(target_length):
            state_input = visited_states[i]
            model_output, hidden_state = decoder(
                state_input, hidden_state)

            loss += criterion(model_output, torch.tensor([selected_actions[i]],
                                                         dtype=torch.long, device=DEVICE))
    return loss.item() / target_length

def validationError(input_data, encoder, decoder, verbose=True):
    """Evalutes the error on a set of input pairs in terms of loss.
    Is intended to be used on a validation or test set to evaluate performance"""
    if encoder !=  None:
        encoder.eval()

    decoder.eval()
    criterion = nn.NLLLoss()
    loss = 0

    for selected_idx in range(len(input_data)):
        pair_loss = evaluate(input_data[selected_idx][0], input_data[selected_idx][1],
                     input_data[selected_idx][2],encoder, decoder, criterion)
        loss += pair_loss

    avg_loss = loss / len(input_data)

    if verbose:
        print('The average validation loss is {:.3} based on {} samples'.format(avg_loss,
len(input_data)))
    return avg_loss

# %% Train supervised model

# Define encoder / decoder models
def train_CartPole_supervised_models():
    """Trains the CartPole supervised model"""
    encoder = None
    encoder_optimizer = None
    decoder = GeneralDecoderRNN(input_size=4, hidden_size=128, output_size=2).to(DEVICE)

    # Set optimizer
    decoder_optimizer = optim.Adam(decoder.parameters())

    n_iterations = 5

    for i in range(n_iterations):
        print('Iteration Number: {}'.format(i))
        trainIters(train_data, encoder, decoder, encoder_optimizer, decoder_optimizer,
                   n_iters=50, print_every=10)

        validationError(val_data[:10], encoder, decoder)

    return encoder, decoder
```

```
def train_FrozenLake_supervised_models():
    """Trains the FrozenLake supervised model"""
    encoder = CNNStateEncoder(128).to(DEVICE)
    encoder_optimizer = optim.Adam(encoder.parameters())
    decoder = GeneralDecoderRNN(input_size=2, hidden_size=128, output_size=4).to(DEVICE)

    # Set optimizer
    decoder_optimizer = optim.Adam(decoder.parameters())

    n_iterations = 5

    for i in range(n_iterations):
        print('Iteration Number: {}'.format(i))
        trainIters(train_data, encoder, decoder, encoder_optimizer, decoder_optimizer,
                   n_iters=2500, print_every=500)

        validationError(val_data[:10], encoder, decoder)

    return encoder, decoder

def supervised_model_reward(input_env, start_state, supervised_encoder, supervised_model):
    """Returns the baseline reward of running the supervised model"""
    supervised_env = deepcopy(input_env)
    supervised_env.state = start_state
    supervised_env.done, supervised_env.ep_reward = False, 0
    state = supervised_env.state

    if supervised_env.name == 'FrozenLake':
        masked_input_map = frozen_lake_env.mask_map(supervised_env.input_map, flatten=False)
        hidden_state = supervised_encoder(masked_input_map).detach()

    else:
        hidden_state = supervised_model.initHidden()

    for i in range(env.max_steps):
        probs, hidden_state = supervised_model(state, hidden_state)

        _, topi = probs.data.topk(1)
        action = topi.squeeze().item()

        state, env_reward, done, _ = supervised_env.step(action)

        if done:
            break
    return supervised_env.ep_reward

#data.save_model(decoder, os.path.join(config.saved_RL_model_path, 'CartPole/CartpoleDecoder.pt'))
#data.save_model(encoder, os.path.join(config.saved_RL_model_path,
'FrozenLake/FrozenLakeEncoder_superexpert.pt'))
#data.save_model(decoder, os.path.join(config.saved_RL_model_path,
'FrozenLake/FrozenLakeDecoder_superexpert.pt'))

#%% Define reinforcement learning models

class RLActor(nn.Module):
    """Defines the RL Actor model"""
    def __init__(self, input_size, hidden_size, output_size):
        super(RLActor, self).__init__()
        self.name = 'RL actor model'
        self.hidden_size = hidden_size

        self.gru = nn.GRU(input_size, self.hidden_size) # Add state space
        self.out = nn.Linear(self.hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

        self.gamma = 0.9999
        self.saved_action_values = []
        self.rewards = []

    def forward(self, input_state, hidden, temperature=1):
        input_state = torch.tensor(input_state, dtype=torch.float32, device=DEVICE).view(1,1,-1)
        output, hidden = self.gru(input_state, hidden)
        output = self.softmax(self.out(output[0]) / temperature)
        return output, hidden
```

```
    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=DEVICE)

class RLCritic(nn.Module):
    """Defines the RL Critic model"""
    def __init__(self, input_size, hidden_size):
        super(RLCritic, self).__init__()
        self.name = 'RL critic model'
        self.hidden_size = hidden_size

        self.h1 = nn.Linear(input_size, self.hidden_size)
        self.h2 = nn.Linear(self.hidden_size, 1)

        self.saved_state_values = []

    def forward(self, state_input, hidden):
        state_input = torch.tensor(state_input, dtype=torch.float32, device=DEVICE).view(1,1,-1)
        x = torch.cat((state_input[0], hidden[0]), 1)
        x = self.h1(x)
        x = F.relu(x)
        value_score = self.h2(x)
        return value_score

    #optionally use dropout in the network
    #self.dropout = nn.Dropout(p=0.6)
    #x = self.dropout(x)

def select_action(input_state, input_hidden_state, actor_model, critic_model=None,
                  teacher_model=None, K=1, use_MLE=False, MCTS_thresh=0):
    """Applies the model on a given input and hidden state to make a prediction of which action to take
        Can use MLE, MCTS, or sampling to select an action"""
    probs, hidden_state = actor_model(input_state, input_hidden_state)
    m = Categorical(probs)

    # Use MLE instead of sampling distribution
    if use_MLE:
        _, topi = probs.data.topk(1)
        action = topi.squeeze()

    elif torch.max(probs).detach() < MCTS_thresh:
        action, hidden_state, _ = MCTS.UCT_search(
                env, input_state, input_hidden_state, actor_model, critic_model,
                5, env.action_space, 1000)
        action = torch.tensor(action, device=config.DEVICE)

    else:
        action = m.sample()

    actor_model.saved_action_values.append(m.log_prob(action))

    if critic_model != None:
        state_value = critic_model(input_state, input_hidden_state)
        critic_model.saved_state_values.append(state_value)

    if teacher_model != None:
        # Add policy distillation error
        actor_probs, _ = actor_model(input_state, input_hidden_state, K)
        supervised_probs, _ = teacher_model(input_state, input_hidden_state, K)
        KL_error = utils.KL_divergence(actor_probs, supervised_probs, K)
        return action.item(), hidden_state, KL_error.item()

    return action.item(), hidden_state, None

def REINFORCE_update(actor_model, actor_optimizer):
    """Update the model when using REINFORCE instead of Actor-Critic"""
    R = 0
    policy_loss = []
    returns = []

    # Discount the rewards back to present
    for r in actor_model.rewards[::-1]:
        R = r + actor_model.gamma * R
        returns.insert(0, R)

    # Scale the rewards
    returns = torch.tensor(returns)
```

```
    ###returns = (returns - returns.mean()) / (returns.std() + EPS)

    # Calculate the loss
    for log_prob, R in zip(actor_model.saved_action_values, returns):
        policy_loss.append(-log_prob * R)

    # Update network weights
    actor_optimizer.zero_grad()
    policy_loss = torch.cat(policy_loss).sum()
    policy_loss.backward()
    actor_optimizer.step()

    # Clear memory
    del actor_model.rewards[:]
    del actor_model.saved_action_values[:]

def actor_critic_update(actor_model, actor_optimizer, critic_model, critic_optimizer,
                        only_update_critic=False):
    """Update the model when using Actor-Critic"""
    R = 0
    saved_actions = actor_model.saved_action_values
    saved_states = critic_model.saved_state_values
    policy_losses = [] # list to save actor (policy) loss
    value_losses = [] # list to save critic (value) loss
    returns = [] # list to save the true values

    # calculate the true value using rewards returned from the environment
    for r in actor_model.rewards[::-1]:
        # calculate the discounted value
        R = r + actor_model.gamma * R
        returns.insert(0, R)

    # Scale the rewards
    returns = torch.tensor(returns)
    ###returns = (returns - returns.mean()) / (returns.std() + EPS) # scaling reduced performance

    for log_prob, value, R in zip(saved_actions, saved_states, returns):
        advantage = R - value.item()

        # calculate actor (policy) loss
        policy_losses.append(-log_prob * advantage)

        # calculate critic (value) loss using L1 smooth loss
        value_losses.append(F.smooth_l1_loss(value, torch.tensor([R], device=DEVICE)))

    # reset gradients
    actor_optimizer.zero_grad()
    critic_optimizer.zero_grad()

    if only_update_critic:
        loss = torch.stack(value_losses).sum()

        # perform backprop
        loss.backward()
        critic_optimizer.step()

    else:
        # sum up all the values of policy_losses and value_losses
        loss = torch.stack(policy_losses).sum() + torch.stack(value_losses).sum()

        # perform backprop
        loss.backward()
        actor_optimizer.step()
        critic_optimizer.step()

    # reset rewards and action buffer
    del actor_model.rewards[:]
    del actor_model.saved_action_values[:]
    del critic_model.saved_state_values[:]

# %% Define environment, models and transfer weights

def load_FrozenLake_models():
    """Instantiate supervised model and load saved weights (CNN model)"""
    supervised_encoder = CNNStateEncoder(128).to(DEVICE)
    supervised_model = GeneralDecoderRNN(input_size=2, hidden_size=128, output_size=4).to(DEVICE)
```

```
    data.load_model(supervised_encoder, os.path.join(
            config.saved_RL_model_path, args.FROZENLAKE_ENCODER))
    data.load_model(supervised_model, os.path.join(
            config.saved_RL_model_path, args.FROZENLAKE_DECODER))

    return supervised_encoder, supervised_model

def load_CartPole_models():
    """Instantiate supervised model and load saved weights"""
    supervised_encoder = None
    supervised_model = GeneralDecoderRNN(input_size=4, hidden_size=128, output_size=2).to(DEVICE)

    data.load_model(supervised_model, os.path.join(config.saved_RL_model_path, args.CARTPOLE_DECODER))

    return supervised_encoder, supervised_model

def create_environment(env_name):
    """Instantiates the environment with the selected hyperparameters"""
    if env_name == 'FrozenLake':
        input_map = frozen_lake_env.generate_random_map(5)
        env = frozen_lake_env.FrozenLakeEnv(input_map, map_frozen_prob=0.75,
                                            sparse=args.sparse_env, changing_map=True)

    elif env_name == 'CartPole':
        env = cart_pole_env.CartPoleEnv(args.sparse_env)

    else:
        print("Please select one of the following environments: ['FrozenLake', 'CartPole']")

    env.seed(config.SEED)
    return env

def init_actor_critic_models(state_space, action_space, init_critic=True,
                             transfer_weights=True, supervised_model=None):
    """Instantiates the actor and critic models as well as the optimizers"""
    # Define actor and critic
    actor_model = RLActor(input_size=state_space, hidden_size=128, output_size=action_space).to(DEVICE)

    # Transfer weights to actor and set optimizer
    if transfer_weights:
        actor_model.load_state_dict(supervised_model.state_dict())

    actor_optimizer = optim.Adam(actor_model.parameters())

    if init_critic:
        critic_model = RLCritic(input_size=(state_space + actor_model.hidden_size),
                                hidden_size=128).to(DEVICE)
        critic_optimizer = optim.Adam(critic_model.parameters())

    else:
        critic_model = None
        critic_optimizer = None

    return actor_model, critic_model, actor_optimizer, critic_optimizer

#%% Load models and environment

def init_RL_environment(env_name, init_critic=True, transfer_weights=True):
    """Instantiates the RL environment and models through using the subfunctions"""

    assert env_name in ['FrozenLake', 'CartPole'], \
        "Please select one of the following environments: ['FrozenLake', 'CartPole']"

    # Load supervised models
    if env_name == 'FrozenLake':
        supervised_encoder, supervised_model = load_FrozenLake_models()
    elif env_name == 'CartPole':
        supervised_encoder, supervised_model = load_CartPole_models()

    # Create environment
    env = create_environment(env_name)

    # Create environment
    actor_model, critic_model, actor_optimizer, critic_optimizer = init_actor_critic_models(
            state_space=env.state_space, action_space=env.action_space,
```

```
            init_critic=init_critic, transfer_weights=transfer_weights,
supervised_model=supervised_model)

    return supervised_encoder, supervised_model, env, \
            actor_model, critic_model, actor_optimizer, critic_optimizer

class HyperParams(object):
    """Sets the experiment hyperparameters"""
    def __init__(self, env_name, n_episodes):
        assert env_name in ['FrozenLake', 'CartPole'], \
        "Please select one of the following environments: ['FrozenLake', 'CartPole']"

        if env_name == 'CartPole':
            self.print_every = 5
            self.t_weight = utils.EpsilonDecay(1, 0, n_episodes, 'Linear') # only Linear works
            self.K = 5
            self.early_stopping_reward_thresh = 295
            self.early_stopping_n_mean = 3

            self.beta = 0.25
            self.sm_baseline_reward = 250
            self.distillation_n_mean = 20

        elif env_name == 'FrozenLake':
            self.print_every = 20
            self.t_weight = utils.EpsilonDecay(1, 0, n_episodes, 'Linear') # only Linear works
            self.K = 5
            self.early_stopping_reward_thresh = 11
            self.early_stopping_n_mean = 50

            self.beta = 1
            self.sm_baseline_reward = 3.5
            self.distillation_n_mean = 500

#%% Training loop

def train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                    supervised_encoder, supervised_model, teacher_model,
                    use_policy_distillation, update_RL_models, only_update_critic, use_MLE,
MCTS_thresh, n_episodes):
    """Main training loop to train the actor and critic models"""

    for i_episode in range(1, n_episodes+1):
        # Reset environment at beginning of episode
        state, ep_reward, done = env.reset(), 0, False

        ep_env_reward = 0
        ep_KL_penalty = 0
        start_state = deepcopy(state)

        if env.name == 'FrozenLake':
            masked_input_map = frozen_lake_env.mask_map(env.input_map, flatten=False)
            hidden_state = supervised_encoder(masked_input_map).detach()

        else:
            hidden_state = actor_model.initHidden()

        # Play environment until maximum number of steps or environment terminates
        for step_i in range(1, env.max_steps+1):
            # Select action
            action, hidden_state, KL_error = select_action(input_state=state,
input_hidden_state=hidden_state,
                                                           actor_model=actor_model,
critic_model=critic_model,
                                                           teacher_model=teacher_model, K=hp.K,
use_MLE=use_MLE,
                                                           MCTS_thresh=MCTS_thresh)

            # Apply action to environment to transition to next step
            state, env_reward, done, _ = env.step(action)

            # Can optionally use relative rewards to supervised model as reward
            if args.sparse_env and args.relative_rewards and done:
                env_reward = env_reward - supervised_model_reward(
                        env, start_state, supervised_encoder, supervised_model)
```

```
            # Can optionally use policy distillation
            if use_policy_distillation:
                avg_RL_reward = np.mean(saved_RL_model_results.env_rewards[-hp.distillation_n_mean:]) \
                        if len(saved_RL_model_results.env_rewards) > hp.distillation_n_mean else 0

                lambda_value= utils.lambda_value(beta=hp.beta,
sm_baseline_reward=hp.sm_baseline_reward,
                                             avg_rewards=avg_RL_reward)
                KL_penalty = lambda_value * -KL_error

                reward = env_reward + KL_penalty
                ep_env_reward += env_reward
                ep_KL_penalty += KL_penalty

            else:
                reward = env_reward
                ep_reward += reward

            actor_model.rewards.append(reward)

            if done:
                break

        # Update models based on reward performance
        if update_RL_models:
            if critic_model != None:
                actor_critic_update(actor_model, actor_optimizer, critic_model, critic_optimizer,
                                only_update_critic=only_update_critic)
            else:
                REINFORCE_update(actor_model, actor_optimizer)


        if use_policy_distillation:
            saved_RL_model_results.env_rewards.append(ep_env_reward)
            saved_RL_model_results.KL_penalty.append(ep_KL_penalty)

            if args.verbose_training and (i_episode % hp.print_every == 0):
                avg_env_reward = np.mean(saved_RL_model_results.env_rewards[-hp.print_every:])
                avg_KL_penalty = np.mean(saved_RL_model_results.KL_penalty[-hp.print_every:])

                print('Episode {} | Avg env reward: {:.2f} | Avg KL penalty: {:.2f} | Lambda value:
{:.2f}'.format(
                      i_episode, avg_env_reward, avg_KL_penalty, lambda_value))

            early_stopping_value = np.mean(saved_RL_model_results.env_rewards[-
hp.early_stopping_n_mean:]) \
                    if len(saved_RL_model_results.env_rewards) > hp.early_stopping_n_mean else 0
            if (early_stopping_value >= hp.early_stopping_reward_thresh) or (i_episode == n_episodes-
1):

                if args.save_models:
                    saved_RL_model_results.save_top_models(actor_model,
'actor_{:.1f}.pt'.format(early_stopping_value))
                    if args.init_critic:
                        saved_RL_model_results.save_top_models(critic_model,
'critic_{:.1f}.pt'.format(early_stopping_value))
                    saved_RL_model_results.export_rewards('model_performance.txt')
                break

        else:
            saved_RL_model_results.env_rewards.append(ep_reward)

            if args.verbose_training and (i_episode % hp.print_every == 0):
                avg_env_reward = np.mean(saved_RL_model_results.env_rewards[-hp.print_every:])
                print('Episode {} | Average reward: {:.2f}'.format(i_episode, avg_env_reward))

            early_stopping_value = np.mean(saved_RL_model_results.env_rewards[-
hp.early_stopping_n_mean:]) \
                    if len(saved_RL_model_results.env_rewards) > hp.early_stopping_n_mean else 0
            if (early_stopping_value >= hp.early_stopping_reward_thresh) or (i_episode == n_episodes-
1):

                if args.save_models:
                    saved_RL_model_results.save_top_models(actor_model,
'actor_{:.1f}.pt'.format(early_stopping_value))
                    if args.init_critic:
                        saved_RL_model_results.save_top_models(critic_model,
'critic_{:.1f}.pt'.format(early_stopping_value))
```

```
                    saved_RL_model_results.export_rewards('model_performance.txt')
                break

def load_RL_models(folder_name, actor_file_name='best', critic_file_name='best'):
    """Instantiate RL models and load trained weights"""
    actor_model = RLActor(input_size=env.state_space, hidden_size=128,
output_size=env.action_space).to(DEVICE)

    if actor_file_name == 'best':
        actor_file_name = 'actor_{:.1f}.pt'.format(
                data.get_top_n_models(
                        os.path.join(config.saved_RL_model_path, args.env_name, folder_name), 'actor',
n=1)[0])

    data.load_model(actor_model, os.path.join(config.saved_RL_model_path, args.env_name,
                                              folder_name, actor_file_name))

    if args.init_critic:
        critic_model = RLCritic(input_size=(env.state_space + actor_model.hidden_size),
                                    hidden_size=128).to(DEVICE)

        if critic_file_name == 'best':
            critic_file_name = 'critic_{:.1f}.pt'.format(
                    data.get_top_n_models(
                            os.path.join(config.saved_RL_model_path, args.env_name, folder_name),
'critic', n=1)[0])

        data.load_model(critic_model, os.path.join(config.saved_RL_model_path, args.env_name,
                                                   folder_name, critic_file_name))

        return actor_model, critic_model

    else:
        return actor_model, None

def load_pretrained_critic(env_name):
    """Intantial RL critic and load trained weights"""
    if env_name =='CartPole':
        data.load_model(critic_model, os.path.join(config.saved_RL_model_path, env_name,
                                                   args.CP_PRETRAINED_CRITIC))
    elif env_name =='FrozenLake':
        data.load_model(critic_model, os.path.join(config.saved_RL_model_path, env_name,
                                                   args.FL_PRETRAINED_CRITIC))
    else:
        print("Please select one of the following environments: ['FrozenLake', 'CartPole']")

#%% Train and Evaluate Model

if args.train_models:
    """Instantiates models and environment, trains and evaluates the model"""

    # Instantiate models and environment
    supervised_encoder, supervised_model, env, \
            actor_model, critic_model, actor_optimizer, critic_optimizer = init_RL_environment(
                    env_name=args.env_name, init_critic=args.init_critic,
transfer_weights=args.transfer_weights)

    # Create folder if saving models
    if args.save_models:
        saved_RL_model_results.init_folder(args, actor_model, critic_model)

    # Optionally load trained models
    if args.load_models:
        actor_model, critic_model = load_RL_models(args.load_model_folder_name)

    # Instantiate teacher model if using policy distillation
    if args.use_policy_distillation:
        teacher_model = TeacherRNN(input_size=env.state_space, hidden_size=128,
                                   output_size=env.action_space).to(DEVICE)
        teacher_model.load_state_dict(supervised_model.state_dict())
    else:
        teacher_model = None

    # Load pretrained critic
    if args.use_pretrained_critic and critic_model is not None:
        load_pretrained_critic(args.env_name)
```

```
    # Optionally pretrain critic
    if args.pretrain_critic_n_episodes > 0:
        hp = HyperParams(args.env_name, args.pretrain_critic_n_episodes)

        train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                        supervised_encoder, supervised_model, teacher_model,
                        use_policy_distillation=False, update_RL_models=True,
                        only_update_critic=True, use_MLE=False, MCTS_thresh=0,
                        n_episodes=args.pretrain_critic_n_episodes)

    # Instantiate hyperparameters
    hp = HyperParams(args.env_name, args.n_episodes)

    # Train models
    train_RL_models(actor_model, critic_model, actor_optimizer, critic_optimizer,
                        supervised_encoder, supervised_model, teacher_model,
                        use_policy_distillation=args.use_policy_distillation,
                        update_RL_models=args.update_RL_models,
                        only_update_critic=False, use_MLE=args.use_MLE,
                        MCTS_thresh=args.MCTS_thresh, n_episodes=args.n_episodes)
```

9.7 ParaPhrasee Env

```
"""Defines environment dynamics for paraphrase generation task as RL problem"""

import torch

import numpy as np
import random
import os

import config
import data
import model_evaluation
import supervised_model as sm
from train_ESIM import load_ESIM_model

DEVICE = config.DEVICE

MAX_LENGTH = config.MAX_LENGTH

SOS_token = config.SOS_token
EOS_token = config.EOS_token

train_pairs = data.TRAIN_PAIRS
val_pairs = data.VAL_PAIRS
test_pairs = data.TEST_PAIRS
vocab_index = data.VOCAB_INDEX

#%%

class ParaPhraseeEnvironment(object):
    """Define the paraphrase generation task in the style of OpenAI Gym"""
    def __init__(self, source_sentence, target_sentence, supervised_encoder,
                 reward_function, similarity_model_name, sentence_pairs):
        self.name = 'ParaPhrasee'
        self.source_sentence = source_sentence # Stored as string
        self.target_sentence = target_sentence # Stored as string
        self.predicted_words = []
        self.reward_function = reward_function # String ex. BLEU
        self.similarity_model_name = similarity_model_name
        self.ESIM_model_name = 'ESIM_noisy_3'
        self.similarity_model, self.fluency_model, self.ESIM_model, \
        self.logr_model, self.std_scaler, \
        self.similarity_dist, self.fluency_dist, self.ESIM_dist = model_evaluation.init_eval_models(
                reward_function=self.reward_function, similarity_model_name=self.similarity_model_name,
                ESIM_model_name=self.ESIM_model_name)
        self.sentence_pairs = sentence_pairs
        self.supervised_encoder = supervised_encoder

        self.max_length = MAX_LENGTH
        self.max_steps = self.max_length
```

```
        self.done = 0
        self.ep_reward = 0
        self.gamma = 0.999
        self.changing_input = True


        self.action_tensor = torch.tensor([[SOS_token]], device=DEVICE)
        self.encoder_outputs = torch.zeros(MAX_LENGTH, supervised_encoder.hidden_size, device=DEVICE)
        self.context, _, _ = sm.embed_input_sentence([self.source_sentence, self.target_sentence],
supervised_encoder,
                                               max_length=self.max_length)
        self.state = (self.action_tensor, self.context)
        self.action_space = vocab_index.n_words

    def pred_sentence(self):
        """Returns the sentence prediction from the environment"""
        output_sentence = ' '.join(self.predicted_words)
        return output_sentence

    def supervised_baseline(self, supervised_decoder):
        """Returns the supervised model prediction for the same sentence for comparative purposes"""
        supervised_decoder_pred, _, baseline_reward = sm.validationMetricPerformance(
                input_pairs=[(self.source_sentence, self.target_sentence)],
encoder=self.supervised_encoder,
                decoder=supervised_decoder, similarity_model=self.similarity_model,
fluency_model=self.fluency_model,
                ESIM_model=self.ESIM_model, logr_model=self.logr_model, std_scaler=self.std_scaler,
                similarity_dist=self.similarity_dist, fluency_dist=self.fluency_dist,
ESIM_dist=self.ESIM_dist,
                vocab_index=vocab_index, verbose=False, metric=self.reward_function)

        supervised_decoder_pred = supervised_decoder_pred[0][1]

        return supervised_decoder_pred, np.around(baseline_reward, 3)

    def step(self, action, decoder_hidden):
        """Key function which represents the transition dynamics.
        given an action (word choice) this returns the updated state FROM THE AGENT
        is effectively the decoder

        All this is effectively doing is checking if the episode is over and returning the
        appropirate reward, else updating the state based on the decoder outputs"""

        # Check whether episode is over
        if (action == EOS_token) or (len(self.predicted_words)>= self.max_length):
            self.state = action, decoder_hidden
            RL_model_reward = model_evaluation.performance_metrics(
                    target_sentence=self.target_sentence, pred_sentence=self.pred_sentence(),
                    similarity_model=self.similarity_model, fluency_model=self.fluency_model,
ESIM_model=self.ESIM_model,
                    logr_model=self.logr_model, std_scaler=self.std_scaler,
                    similarity_dist=self.similarity_dist, fluency_dist=self.fluency_dist,
ESIM_dist=self.ESIM_dist,
                    vocab_index=vocab_index, metric=self.reward_function)
            # Calculate relative reward
            self.ep_reward = np.around(RL_model_reward, 3)
            self.done = 1
        else:
            self.state = action, decoder_hidden

            # Add word to pred words
            self.predicted_words.append(vocab_index.index2word[action.item()])

        return self.state, self.ep_reward, self.done, None

    def reset(self):
        """Resets the environment to a random initial state through picking a random sentence from the
            sentence input pairs"""
        if self.changing_input:
            sentence_pair = random.choice(self.sentence_pairs)
            self.source_sentence = sentence_pair[0] # Stored as string
            self.target_sentence = sentence_pair[1] # Stored as string

        self.predicted_words = []

        self.ep_reward = 0
```

```
        self.done = 0

        self.action_tensor = torch.tensor([[SOS_token]], device=DEVICE)
        self.encoder_outputs = torch.zeros(MAX_LENGTH, self.supervised_encoder.hidden_size,
device=DEVICE)
        self.context, _, _ = sm.embed_input_sentence([self.source_sentence, self.target_sentence],
                                             self.supervised_encoder,
max_length=self.max_length)
        self.state = (self.action_tensor, self.context)

        return self.state
```

9.8 Train ESIM

```
"""Defines and trains an ESIM model"""

import torch
import torch.nn.functional as F
import numpy as np
import argparse
import os

from ESIM.ESIM import ESIM

import config
from encoder_models import create_vocab_tensors
import data
from utils import tensorsFromPair

DEVICE = config.DEVICE
GPU_ENABLED = config.GPU_ENABLED
MAX_LENGTH = config.MAX_LENGTH

SOS_token = config.SOS_token
EOS_token = config.EOS_token

# Load Data
if __name__ == '__main__':
    train_pairs = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_train_pairs.npy'))
    y_train = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_train_labels.npy'))
    y_train = torch.from_numpy(y_train).to(DEVICE)

    val_pairs = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_val_pairs.npy'))
    y_val = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_val_labels.npy'))
    y_val = torch.from_numpy(y_val).to(DEVICE)

    test_pairs = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_test_pairs.npy'))
    y_test = data.load_np_data(os.path.join(config.saved_ESIM_model_path, 'aug_test_labels.npy'))
    y_test = torch.from_numpy(y_test).to(DEVICE)

vocab_index = data.VOCAB_INDEX

parser = argparse.ArgumentParser(description='Train_Supervised_Model')
parser.add_argument('--train_models', action='store_true',
                    help='enable training of models')
parser.add_argument('--folder_name', type=str,
                    help='Brief description of experiment (no spaces)')

parser.add_argument('--n_epochs', type=int, default=1,
                    help='number of epochs to train online in each loop (default: 1)')
parser.add_argument('--verbose_training', action='store_true',
                    help='print results during training')

parser.add_argument('--load_models', action='store_true', help='Load pretrained model from prior
point')
parser.add_argument('--load_model_folder_name', type=str,
                    help='folder which contains the saved models to be used')

if __name__ == '__main__':
    args = parser.parse_args()
    args.save_models = 0

    if args.train_models:
        args.save_models = 1
        saved_ESIM_model_results = data.SaveSupervisedModelResults(args.folder_name)
```

```
        saved_ESIM_model_results.check_folder_exists()

pretrained_emb_file = 'pretrained_emb_100k.npy'


#%% Manual commands for testing

#args.train_models = 1
#args.verbose_training = 1
#saved_ESIM_model_results = data.SaveSupervisedModelResults('ESIM')

#%%

def mask_batch(input_batch_pairs):
    """Convert batch of sentence pairs to tensors and masks for ESIM model"""
    input_tensor = torch.zeros((MAX_LENGTH, len(input_batch_pairs)), dtype=torch.long, device=DEVICE)
    target_tensor = torch.zeros((MAX_LENGTH,len(input_batch_pairs)), dtype=torch.long, device=DEVICE)

    for idx, pair in enumerate(input_batch_pairs):
        encoded_input, encoded_target = tensorsFromPair(pair)
        input_tensor[:len(encoded_input), idx], target_tensor[:len(encoded_target), idx] = \
            encoded_input.view(-1), encoded_target.view(-1)

    input_tensor_mask, target_tensor_mask = input_tensor != 0, target_tensor != 0
    input_tensor_mask, target_tensor_mask = input_tensor_mask.float(), target_tensor_mask.float()

    return input_tensor, input_tensor_mask, target_tensor, target_tensor_mask

def ESIM_pred(input_pairs, model, temperature=1):
    """Returns probability that sentences are paraphrases from trained ESIM model"""
    model.eval()

    input_tensor, input_tensor_mask, target_tensor, target_tensor_mask = mask_batch(input_pairs)

    with torch.no_grad():
        output = model(input_tensor, input_tensor_mask, target_tensor, target_tensor_mask)
        probs = F.softmax(output / temperature, dim=1)

    return probs[:,1]

def validation_error(val_pairs, y_val, model, temperature=1, batch_size=32, verbose=True):
    """Evalutes the error on a set of input pairs in terms of loss.
    Is intended to be used on a validation or test set to evaluate performance"""
    model.eval()
    total_val_loss = 0
    val_sents_scanned = 0
    val_num_correct = 0
    batch_counter = 0
    batch_size = min(len(val_pairs), batch_size)

    output_probs = torch.zeros((len(val_pairs),2), device=DEVICE)

    for idx in range(len(val_pairs) // batch_size):
        input_tensor, input_tensor_mask, target_tensor, target_tensor_mask = mask_batch(
                val_pairs[idx*batch_size:(idx+1)*batch_size])
        batch_labels = y_val[idx*batch_size:(idx+1)*batch_size]

        with torch.no_grad():
            output = model(input_tensor, input_tensor_mask, target_tensor, target_tensor_mask)
            probs = F.softmax(output / temperature, dim=1)

        loss = criterion(output, batch_labels)

        output_probs[idx*batch_size:(idx+1)*batch_size,:] = probs

        result = output.detach().cpu().numpy()
        a = np.argmax(result, axis=1)
        b = batch_labels.data.cpu().numpy()

        val_num_correct += np.sum(a == b)
        val_sents_scanned += len(batch_labels)

        batch_counter += 1

        batch_loss = loss.data.item()
        total_val_loss += batch_loss
```

```
    val_loss = total_val_loss / batch_counter
    val_accuracy = (val_num_correct / val_sents_scanned)

    if verbose:
        print('{} batches | validation loss: {:.3} | validation accuracy: {:.3}'.format(
                    batch_counter, val_loss, val_accuracy))

    return val_loss, val_accuracy, output_probs

def model_pipeline(model, criterion, optimizer, batch_size=32, num_epochs=1,
                   report_interval=10, early_stopping_interval=100, verbose=True):
    """Model pipeline which trains model and also generates examples while training and evaluation
    on the validation set for potential early stopping"""
    batch_counter = 0

    print('start training...')
    model.train()

    for epoch in range(num_epochs):
        model.train()
        print('--' * 20)
        train_sents_scanned = 0
        train_num_correct = 0
        batch_counter = 0

        for idx in range(len(train_pairs) // batch_size):
            input_tensor, input_tensor_mask, target_tensor, target_tensor_mask = mask_batch(
                    train_pairs[idx*batch_size:(idx+1)*batch_size])
            batch_labels = y_train[idx*batch_size:(idx+1)*batch_size]

            optimizer.zero_grad()
            output = model(input_tensor, input_tensor_mask, target_tensor, target_tensor_mask)
            loss = criterion(output, batch_labels)
            loss.backward()

            result = output.detach().cpu().numpy()
            a = np.argmax(result, axis=1)
            b = batch_labels.data.cpu().numpy()

            train_num_correct += np.sum(a == b)
            train_sents_scanned += len(batch_labels)

            optimizer.step()
            training_loss = loss.detach().item()
            batch_counter += 1

            saved_ESIM_model_results.train_loss.append(np.around(training_loss, 4))

            if batch_counter % report_interval == 0 and verbose == True:
                print('{} epochs, {} batches | training batch loss: {:.3} | train accuracy:
{:.3}'.format(
                        epoch, batch_counter, training_loss, train_num_correct / train_sents_scanned))

            if batch_counter % early_stopping_interval == 0:
                val_prop = int(0.05 * len(val_pairs))
                random_idx = np.random.choice(val_prop, val_prop, replace=False)
                sample_val_pairs, sample_y_train = val_pairs[random_idx], y_val[random_idx]

                val_loss, val_accuracy, _ = validation_error(
                        sample_val_pairs, sample_y_train, model,
                        temperature=1, batch_size=32, verbose=verbose)
                model.train()

                saved_ESIM_model_results.val_loss.append(np.around(val_accuracy, 4))


        saved_ESIM_model_results.save_top_models(model, 'ESIM_{:.3f}.pt'.format(
                val_accuracy))
    saved_ESIM_model_results.export_loss('training_loss.txt', 'val_loss.txt')

class HyperParams(object):
    """Sets the experiment hyperparameters"""
    def __init__(self, print_every=10):
        self.print_every = print_every
        self.early_stopping_interval = 150
```

```
        self.dim_word = 300
        self.batch_size = 32
        self.n_words = vocab_index.n_words
        self.n_classes = 2

def load_pretrained_emb(input_path=None):
    """Loads word embeddings for vocabulary or creates new vocabulary"""
    if input_path is not None:
        return data.load_np_data(input_path)
    else:
        return create_vocab_tensors(vocab_index)[0].cpu().numpy()

def load_ESIM_model(folder_name, file_name='best', path_override=None):
    """Instantiates and loads ESIM model"""
    hp = HyperParams()

    pretrained_emb = load_pretrained_emb(os.path.join(config.saved_ESIM_model_path,
pretrained_emb_file))
    ESIM_model = ESIM(hp.dim_word, hp.n_classes, hp.n_words, hp.dim_word, pretrained_emb).to(DEVICE)


    if path_override is not None:
        data.load_model(ESIM_model, os.path.join(path_override, file_name))

    else:
        if file_name == 'best':
            file_name = 'ESIM_{:.3f}.pt'.format(data.get_top_n_models(
                    os.path.join(config.saved_ESIM_model_path, folder_name), 'ESIM', n=1,
descending=True)[0])

        data.load_model(ESIM_model, os.path.join(config.saved_ESIM_model_path, folder_name, file_name))

    return ESIM_model

#%%

class RLAdversary():
    """Defines RL adversary model for use as reward function"""
    def __init__(self, folder_name, file_name='best'):
        super(RLAdversary, self).__init__()
        self.name = 'ESIM RL Adversary'
        self.folder_name = folder_name
        self.file_name = file_name
        self.model, self.criterion, self.optimizer = self.init_model()

        self.pred_pairs = []
        self.target_pairs = []

        self.batch_size = 32
        self.num_epochs = 1

        self.update_iter = 0
        self.training_accuracy = {}

    def init_model(self):
        model = load_ESIM_model(self.folder_name, self.file_name)

        criterion = torch.nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters())

        return model, criterion, optimizer

    def create_update_data(self):
        update_pairs = np.concatenate([self.pred_pairs, self.target_pairs])
        y_update = torch.zeros((len(update_pairs),), dtype=torch.long, device=DEVICE)
        y_update[len(self.pred_pairs):] = 1

        random_idx = np.random.choice(len(y_update), len(y_update), replace=False)
        update_pairs, y_update = update_pairs[random_idx], y_update[random_idx]
        return update_pairs, y_update

    def update_model(self):
        self.update_iter += 1
        self.training_accuracy[self.update_iter] = []

        update_pairs, y_update = self.create_update_data()
```

```
        self.model.train()

        batch_size = min(len(update_pairs), self.batch_size)

        for epoch in range(self.num_epochs):
            self.model.train()
            train_sents_scanned = 0
            train_num_correct = 0

            for idx in range(len(update_pairs) // batch_size):
                input_tensor, input_tensor_mask, target_tensor, target_tensor_mask = mask_batch(
                        update_pairs[idx*batch_size:(idx+1)*batch_size])
                batch_labels = y_update[idx*batch_size:(idx+1)*batch_size]

                self.optimizer.zero_grad()
                output = self.model(input_tensor, input_tensor_mask, target_tensor, target_tensor_mask)
                loss = self.criterion(output, batch_labels)
                loss.backward()

                self.optimizer.step()

                result = output.detach().cpu().numpy()
                a = np.argmax(result, axis=1)
                b = batch_labels.data.cpu().numpy()

                train_num_correct += np.sum(a == b)
                train_sents_scanned += len(batch_labels)

                self.training_accuracy[self.update_iter].append(train_num_correct /
train_sents_scanned)

        self.pred_pairs = []
        self.target_pairs = []

    def reset(self):
        self.model, self.criterion, self.optimizer = self.init_model()

        self.pred_pairs = []
        self.target_pairs = []

        self.batch_size=32
        self.num_epochs=1

        self.update_iter = 0
        self.training_accuracy = {}

#%%

if (__name__ == '__main__') and args.train_models:
    """Initializes models subject to cmd line args and then trains and evaluates performance"""

    # Set the hyperparameters
    hp = HyperParams()

    # Load pretrained embeddings and model
    pretrained_emb = load_pretrained_emb(os.path.join(config.saved_ESIM_model_path,
pretrained_emb_file))
    model = ESIM(hp.dim_word, hp.n_classes, hp.n_words, hp.dim_word, pretrained_emb).to(DEVICE)

    # Set criterion and optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())

    # Create folder if saving
    if args.save_models:
        saved_ESIM_model_results.init_folder(args, None, None)

    # Train model
    model_pipeline(model=model, criterion=criterion, optimizer=optimizer, batch_size=hp.batch_size,
                   num_epochs=args.n_epochs, report_interval=hp.print_every,
                   early_stopping_interval = hp.early_stopping_interval, verbose=args.verbose_training)
```

9.9 MCTS

```
"""Monte Carlo Tree Search implementation for both FrozenLake and ParaPhrasee environments"""
"""Source: https://github.com/brilee/python_uct/blob/master/numpy_impl.py"""

import collections
import numpy as np
import math

import torch
from torch.distributions import Categorical
from copy import deepcopy

import data
import config
import model_evaluation

DEVICE = config.DEVICE

#Load data
vocab_index = data.VOCAB_INDEX

# Define environment wrappers as layer between MCTS code and full environments

class ParaPhraseeEnvWrapper():
    def __init__(self, input_env):
        self.env = deepcopy(input_env)
        self.env.max_length = 11
        self.max_steps = 11

    def take_action(self, input_state, action):
        self.env.state = input_state
        state, _, terminal, _ = self.env.step(torch.tensor(action, device=DEVICE), self.env.state[1])
        self.env.done = False
        return state, terminal

    def get_reward(self):
        return model_evaluation.performance_metrics(
                target_sentence=self.env.target_sentence, pred_sentence=self.env.pred_sentence(),
                similarity_model=self.env.similarity_model, fluency_model=self.env.fluency_model,
                ESIM_model=self.env.ESIM_model, logr_model=self.env.logr_model,
std_scaler=self.env.std_scaler,
                similarity_dist=self.env.similarity_dist, fluency_dist=self.env.fluency_dist,
                ESIM_dist=self.env.ESIM_dist, vocab_index=vocab_index, metric=self.env.reward_function)

#input_map = frozen_lake_env.generate_random_map(5)
#env = frozen_lake_env.FrozenLakeEnv(input_map, map_frozen_prob=0.75, changing_map=False)
#
def FrozenLake_get_reward(input_map, state):
    if input_map[state] == 'G':
        return 20
    elif input_map[state] == 'H':
        return -10
    else:
        return -1

class FrozenLakeEnvWrapper():
    def __init__(self, input_env):
        self.env = deepcopy(input_env)
        self.max_steps = 25

    def take_action(self, input_state, action):
        self.env.state = input_state
        state, _, terminal, _ = self.env.step(action)
        self.env.done = False
        return state, terminal

    def get_reward(self, input_state):
        return FrozenLake_get_reward(self.env.input_map, input_state)

#%% MCTS implementation - Builds out search tree """Based heavily on:
https://github.com/brilee/python_uct/blob/master/numpy_impl.py"""

class UCTNode():
    """Defines nodes and their properties, as well as code handling the construction of the tree.
```

```
        Uses a vectorized implementation in Numpy to improve speed"""
    def __init__(self, state, hidden_state, move, action_space, parent=None, terminal=False):
        self.state = state
        self.hidden_state = hidden_state
        self.move = move
        self.action_space = action_space
        self.is_expanded = False
        self.parent = parent    # Optional[UCTNode]
        self.children = {}    # Dict[move, UCTNode]
        self.child_probs = np.zeros([self.action_space], dtype=np.float32)
        self.child_total_value = np.zeros([self.action_space], dtype=np.float32)
        self.child_number_visits = np.zeros([self.action_space], dtype=np.float32)

        # Update terminal condition based on env
        #self.terminal = True if self.move == config.EOS_token else False # For ParaPhrasee
        self.terminal = terminal

    @property
    def number_visits(self):
        return self.parent.child_number_visits[self.move]

    @number_visits.setter
    def number_visits(self, value):
        self.parent.child_number_visits[self.move] = value

    @property
    def total_value(self):
        return self.parent.child_total_value[self.move]

    @total_value.setter
    def total_value(self, value):
        self.parent.child_total_value[self.move] = value

    def child_Q(self):
        """Adjusts the value based on the number of visits"""
        return self.child_total_value / (0.01 + self.child_number_visits)

    def child_U(self):
        """Calculates the score for determining which node should be explored"""
        return math.sqrt(self.number_visits) * (
                self.child_probs / (0.01 + torch.tensor(self.child_number_visits,
device=config.DEVICE)))

    def best_child(self):
        """Selects the best child node as main"""
        return torch.argmax(torch.tensor(self.child_Q(), device=config.DEVICE) + self.child_U()).item()

    def select_leaf(self, env_wrapper, actor_model):
        current = self
        while current.is_expanded:
            best_move = current.best_child()
            current = current.maybe_add_child(env_wrapper, best_move, actor_model)
        return current

    def expand(self, child_probs):
        self.is_expanded = True
        self.child_probs = child_probs

    def maybe_add_child(self, env_wrapper, move, actor_model):
        """Explores the tree and expands as needed when reaching an unexplored child"""
        if env_wrapper.env.name == 'ParaPhrasee':
            if move not in self.children:
                state, terminal = env_wrapper.take_action((self.state, self.hidden_state), move)
                _, hidden_state = actor_model(self.state, self.hidden_state)

                self.children[move] = UCTNode(
                        state[0], hidden_state, move, self.action_space, parent=self,
terminal=terminal)
            return self.children[move]

        elif env_wrapper.env.name == 'FrozenLake':
            if move not in self.children:
                state, terminal = env_wrapper.take_action(self.state, move)
                _, hidden_state = actor_model(self.state, self.hidden_state)

                self.children[move] = UCTNode(
```

```
                        state, hidden_state, move, self.action_space, parent=self, terminal=terminal)
            return self.children[move]
        else:
            print('Select either ParaPhrasee or FrozenLake env')

    def backup(self, value_estimate: float):
        """Propogates the estimated score back to the relevant nodes"""
        current = self
        while current.parent is not None:
            current.number_visits += 1
            current.total_value += (value_estimate)
            current = current.parent

class DummyNode(object):
    """Defines empty node to be used as root"""
    def __init__(self):
        self.parent = None
        self.child_total_value = collections.defaultdict(float)
        self.child_number_visits = collections.defaultdict(float)

def sample_rollout(input_env, actor_model, temperature, input_state, input_hidden_state, max_steps):
    """Randomly uses rollout until reaching terminal node rather than using NN to approximate value"""
    rollout_env = deepcopy(input_env)
    rollout_env.state = input_state

    state = rollout_env.state
    hidden_state = input_hidden_state

    ep_env_reward = 0
    #selected_actions = []

    for step_i in range(max_steps):
        probs, hidden_state = actor_model(state, hidden_state, temperature)
        m = Categorical(probs)
        action = m.sample().item()

        state, env_reward, done, _ = rollout_env.step(action)
        ep_env_reward += env_reward
        #selected_actions.append(action)

        if done:
                break

    return ep_env_reward

def UCT_search(env, input_state, hidden_state,
               actor_model, critic_model, temperature, action_space, n_iters):
    """Main function which runs the pipeline for a given root / starting state to
        produce the MCTS prediction"""
    if env.name == 'ParaPhrasee':
        env_wrapper = ParaPhraseeEnvWrapper(env)
    elif env.name == 'FrozenLake':
        env_wrapper = FrozenLakeEnvWrapper(env)
    else:
        print('Select either ParaPhrasee or FrozenLake env')

    root = UCTNode(input_state, hidden_state, move=None, action_space=action_space,
                   parent=DummyNode(), terminal=False)

    for _ in range(n_iters):
        leaf = root.select_leaf(env_wrapper, actor_model)

        child_probs = actor_model(leaf.state, leaf.hidden_state, temperature)[0].detach()[0]
        if leaf.terminal:
            if env_wrapper.env.name == 'ParaPhrasee':
                value_estimate = env_wrapper.get_reward()
            elif env_wrapper.env.name == 'FrozenLake':
                value_estimate = env_wrapper.get_reward(leaf.state)
            else:
                print('Select either ParaPhrasee or FrozenLake env')
        else:
            if critic_model is not None:
                value_estimate = critic_model(leaf.state, leaf.hidden_state).detach().item()
            else:
                value_estimate = sample_rollout(env_wrapper.env, actor_model, temperature, leaf.state,
                                                leaf.hidden_state, env_wrapper.max_steps)
```

```
    leaf.expand(child_probs)
    leaf.backup(value_estimate)

MCTS_action = np.argmax(root.child_number_visits)
MCTS_hidden_state = root.children[MCTS_action].hidden_state

return MCTS_action, MCTS_hidden_state, root
```