REINFORCEMENT LEARNING FOR TEXT CLASSIFICATION: AN EVALUATION OF POLICY-GRADIENT METHODS WITH VARIOUS TOPOLOGIES

by
EMRE BATUHAN BALOĞLU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2023

REINFORCEMENT LEARNING FOR TEXT CLASSIFICATION: AN EVALUATION OF POLICY-GRADIENT METHODS WITH VARIOUS TOPOLOGIES

Approved by:

Prof. Dr. Berrin Yanıkoğlu .....
(Thesis Supervisor)

Asst. Prof. Dr. Onur Varol .....

Asst. Prof. Dr. Vahid Tavakol Aghaei .....

Date of Approval: July 26, 2023

ABSTRACT

REINFORCEMENT LEARNING FOR TEXT CLASSIFICATION: AN EVALUATION OF POLICY-GRADIENT METHODS WITH VARIOUS TOPOLOGIES

EMRE BATUHAN BALOĞLU

Computer Science and Engineering M.S. THESIS, JULY 2023

Thesis Supervisor: Prof. Dr. Berrin Yanıkoğlu

Keywords: reinforcement learning, natural language processing, text classification

Usage of reinforcement learning (RL) in natural language processing (NLP) tasks has gained momentum in recent years. In this thesis, we present an improved approach to the task of text classification through the integration of various deep learning topologies such as transformers and large language models (LLMs) into the feature extraction process within a reinforcement learning framework. In this proposed method, the RL policies are trained to observe a portion of the text and determine whether to classify the text or to proceed to the next part of the document. The policies were optimized with the REINFORCE (Williams, 1992) algorithm utilizing a designed reward signal. The effectiveness of the proposed method was evaluated and compared against other state-of-the-art models on standard text classification benchmark datasets, demonstrating the superiority of the proposed approach in terms of efficiency while losing little performance in accuracy. The results indicate that the use of the LLMs in the feature extraction process, coupled with RL policies with designed reward signals, provides a promising avenue for the development of effective and efficient text classification models.

ÖZET

METİN SINIFLANDIRMASI İÇİN PEKİŞTİRMELİ ÖĞRENME: POLİTİKA-GRADYAN METOTLARININ FARKLI TOPOLOJİLER ÜZERİNDE DEĞERLENDİRİLMESİ

EMRE BATUHAN BALOĞLU

Bilgisayar Bilimi ve Mühendisliği YÜKSEK LİSANS TEZİ, TEMMUZ 2023

Tez Danışmanı: Prof. Dr. Berrin Yanıkoğlu

Anahtar Kelimeler: pekiştirmeli öğrenme, doğal dil işleme, metin sınıflandırma

Pekiştirmeli öğrenmenin (RL) doğal dil işleme (NLP) görevlerindeki kullanımı son yıllarda hız kazanmıştır. Bu tezde, dönüştürücüler (transformer) ve büyük dil modelleri (LLM'ler) gibi çeşitli derin öğrenme topolojilerinin, özellik çıkarma sürecine bir pekiştirmeli öğrenme çerçevesi içinde entegre edilmesiyle, metin sınıflandırma problemine geliştirilmiş bir yaklaşım sunuyoruz. Önerilen yöntemde, pekiştirmeli öğrenme politikaları, bir metin bölümünü gözlemlemek ve metni sınıflandırmak veya metnin bir sonraki bölümüne geçmek konusunda karar vermek için eğitilir. Politikalar, REINFORCE (Williams, 1992) algoritmasıyla optimize edilir ve bunun için tasarlanmış bir ödül sinyali kullanılır. Önerilen yöntemin etkinliği, standart metin sınıflandırma veri kümeleri üzerinde, diğer güncel modellerle karşılaştırılarak değerlendirildi ve önerilen yaklaşımın verimlilik açısından üstünlüğü ve tutarlılık açısından küçük bir performans kaybı gösterdiği görülmüştür. Sonuçlar, büyük dil modellerinin özellik çıkarma sürecinde kullanılması ve tasarlanmış ödül sinyali ile pekiştirmeli öğrenme politikalarının birleştirilmesinin etkili ve verimli metin sınıflandırma modellerinin geliştirilmesi için umut verici bir yol sağladığını göstermektedir.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and appreciation to every individual that have supported the completion of my master thesis. Without their guidance, encouragement, and assistance, this work would not have been possible.

First and foremost, I am deeply indebted to my thesis supervisor, Berrin Yanıkoğlu, for her invaluable guidance and mentorship throughout this research endeavor. Her expertise, patience, and steadfast support have been irreplaceable in shaping and refining this thesis. I am truly grateful for her constructive feedback and insightful suggestions.

I would like to extend my gratitude to the members of my thesis committee, Onur Varol and Vahid Tavakol Aghaei, for their time, expertise, and valuable inputs during the evaluation and review process.

I am also thankful to Department of Computer Science in Sabanci University for providing the necessary resources and facilities that enabled me to carry out this study. The well-equipped library, access to research databases, and technical support were vital in gathering relevant information and conducting the required analysis.

I would like to extend my sincere appreciation to the Turkish Scientific and Technological Research Council (TUBITAK) for awarding me the scholarship that funded my master's studies. I am grateful for their recognition of the importance of academic pursuits and their commitment to nurturing future scholars and researchers.

Furthermore, I am indebted to my family for their unwavering love, support, and understanding throughout my academic journey. Their constant encouragement, belief in my abilities, and sacrifices have been the driving force behind my success.

Additionally, I would like to express my sincere appreciation to my friends Kerem Örs, Ozan Can Şahin and Vehbi Kepkep who have been a source of motivation, encouragement, and intellectual stimulation.

In conclusion, I extend my deepest gratitude to all those who have played a part, big or small, in the completion of this master thesis. Their support, encouragement, and assistance have been truly invaluable, and I am humbled by their contributions.

Thank you all.

To Mustafa Kemal Atatürk
Whose ideas guide me in life.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN: convolutional Neural Network

LLM: Large Language Model

LSTM: Long-Short Term Memory

ML: Machine Learning

NLP: Natural Language Processing

RL: Reinforcement Learning

RLHF: Reinforcement Learning from Human Feedback

RNN: Recurrent Neural Network

1. INTRODUCTION

Text classification is one of the essential and challenging tasks for natural language processing (NLP) research, with its ramifications spanning sentiment analysis, spam detection, topic classification, and more. Concurrently, Reinforcement Learning (RL), lauded for its robust generalization capabilities and its alignment with evolutionary neuroscientific paradigms, has found increasing applicability across various facets of machine learning. The parallels between RL's non-stationary environments and the ever-evolving corpus of languages, as well as agents' decision-making processes under uncertainity and the employment of unseen or partially observed texts in NLP, posit reinforcement learning a good approach for addressing NLP challenges (Ramamurthy et al., 2023; Silver et al., 2021).

One of the foremost challenges in NLP today is the development of lightweight and efficient solutions. Such solutions are imperative not only for their intrinsic value but also for their critical application in scenarios like mobile applications, prompt-response customer services, and real-time sentiment classification. Thus, in this thesis, we scrutinize the potential applications of reinforcement learning within the domain of text classification. We intend to reduce the computation cost of models in training and inference time, whilst preserving performance on the conventional classification metrics. Using the study of Yu et al. (2018) as our main inspiration, we explore different approaches and architectures from both RL and NLP domains. We propose new feature extractors and reward signals to achieve this objective and we these architectures on two widely used benchmark datasets.

Emerging research shows success of using reinforcement learning on text classification tasks. For instance, Xu *et al.* (Xu et al., 2019) achieve significant results using policy gradient approach in an adversarial setting for semantic classification task. Moreover, Yang *et al.* (Yang et al., 2018a) show good progress on multi label text classification by formulating the problem as a Markov Decision Process and using an encoder-decoder model with self-critical policy gradient algorithm.

1.1 Related Work on Text Classification with Reinforcement Learning

Although there are few detailed investigations made on this subject, we observe that efforts to combine RL on different machine learning approaches including NLP is growing. Several studies have been performed on using RL on text classification tasks, and on these studies we generally encounter two approaches to incorporate RL for text classification task. One approach is to use RL agents directly as predictors for the task using classification models such as support vector machines or neural networks as state-action parameters, and the other approach is to use RL agents as feature generators/extractors for classification models, again using different models as parameter space.

As one of the first attempts to solve text classification problem with RL, Dulac-Arnold et al. (2011) formulate the problem as a sequential decision-making process with an RL agent, which is a predictor for class labels. Given a sample text, a Markov Decision Process is created where each state is a sentence from the given text, and actions are whether to classify a label to the text or continue to the next sentence or stop. The agent skims through the states sentence-by-sentence deciding an action, starting from the first sentence of the text. The agent is trained through *Approximate Policy Iteration with Rollouts.* (Lagoudakis and Parr, 2003) algorithm, and able to achieve better performance than traditionally trained models.

Following this study, Yu et al. (2018) propose a similar scheme exploiting RNNs with Gated Recurrent Unit (GRU) and policy gradient algorithm (Williams, 1992), additionally adding an action to re-read a given text chunk. Correspondingly, Martinez et al. (2020) make use of Partially-Observable Markov Decision Processes (POMDP), propose strategies of prioritized sampling, prioritized storing and random episode initialization to address the agent's memory imbalances in a similar problem formulation.

In another investigation, Mao et al. (2019) use RL agents as predictors, mainly focusing on hierarchical text-classification, where samples have more than one label given in a tree or directed acyclic graph (DAG). Unlike to the study of Yang et al. (2018a), label order is important in the context of this study. They made use of TextCNN (Kim, 2014), HAN(Yang et al., 2016) and bow-CNN (Johnson and Zhang, 2015), using hidden state parameters of the models as states of a RL agent. Given sample text and labels as a hierarchy in a DAG, starting from the root node, the agent decides whether to assign the current label in the node to the given text, thus generating a sequence of labels. The agent traverses the entire DAG, receiving

F1 score between the generated sequence of labels and the ground truth ones as feedback signal at each time step.

On the other hand, RL is incorporated into text classification on some studies by introducing generative schemes in the model training steps. Xu et al. (2019) utilized an generative-adversarial reinforcement training scheme to achieve robust sentiment classification task. They introduce a long-short term memory (LSTM) based generator, which generates action sequences given any sentence, as an RL agent. Each action sequence has the same length as the given sentence and the agent decides an action; whether to change a word with one of its synonym, subordinate, superior or neighbor word or introduce no change. Through this, the agent aims to confuse the classifier, receiving a positive or negative feedback signal based on the predictions of the classifier. Here, when the classifier makes a correct prediction, the agent receives a negative reward, and conversely the agent receives a positive reward when the classifier makes an incorrect prediction. This creates an generative-adversarial training scheme improving the performance of the classifiers.

Likewise, Chai et al. (2020) employed another generative design, intending to generate descriptions for the labels that models differentiate. Main architecture used in this design is BERT (Devlin et al., 2018), which is both used to generate these descriptions and for text representation for classifier models. They suggested that given proper descriptions, improvements will be observed on model performances. They apply three methods to generate descriptions for labels: Firstly, preparation of hand-crafted descriptions by area experts, which is highly labor intensive and suboptimal. Secondly, generating a substring of a given text, by using an RL agent that decides start and end indexes of the substring. Lastly, generating a sequence of words with another RL agent, where the action of the agent is the generation of words in a sequence-to-sequence (seq2seq) manner. The agents on both of these schemes receive the probability of models' making a correct prediction as a feedback signal from the classifier as a reward, where the classifiers get the BERT representation of the text concatenated to label descriptions for each label.

Moreover, Yang et al. (2018a) addressed the multi-label classification problem, where samples can have more than one label. They refer to the major challenges on this task, that are exploiting the internal correlations of the labels and heeding the swapping invariance problem, when order of the labels is unimportant. Main approach they give in their study was to use an Encoder-Decoder architecture consisting of one Bidirectional-LSTM layer as an encoder, and two unidirectional LSTM layers as decoders, latter acting as an RL agent. The agent generates a sequence of labels, and gets a feedback signal as the F1 score between the generated labels and the

real labels of the sample. F1 score provides the flexibility of swapping invariance to some degree, while other parts of the encoder-decoder structure captures the internal correlations of the labels.

Furthermore, Wang et al. (2019) focus on document-level aspect sentiment classification. They build a hierarchical RL scheme inspired by human cognitive process. They use two RL agents for selecting aspect-relevant clauses in given text, and for selecting sentiment-relevant words from the clauses that the first agent generates. Then, a classifier takes the outputs of these agents as selected clauses and words, and provides a feedback signal for them, based on the performance of this classifier.

In summary, different approaches have been employed for text classification tasks. We see that generative models trained with an RL scheme tend to enhance the model performance as in Xu et al. (2019), Chai et al. (2020), Yang et al. (2018b), Wang et al. (2019) in classification tasks, while some RL agents as predictors can also perform well as in Mao et al. (2019).

1.2 Reinforcement Learning on Classification and other NLP Tasks

Due to the scarcity of research in applications of RL in text classification, we also review the research in broader applications of RL in classification tasks. RL has been considered a favorable candidate for classification tasks for at least some decades (Lagoudakis and Parr, 2003), (Wiering et al., 2011) and is posited on various classification tasks such as vehicle classification from images (Zhao et al., 2017), 3D medical image classification (Zhu et al., 2022), emotion classification from face images Li and Xu (2020), video anomaly detection (Mansour et al., 2021) and class imbalanced classification (Lin et al., 2020).

Reinforcement Learning has been attempted to be used in other NLP tasks as well. Satija and Pineau (2016) and Xia et al. (2016) and Wu et al. (2018) perform a general study of RL training on neural machine translation (NMT) task, while Qian et al. (2018) leverages RL on NMT task in a multimodal setting. Kreutzer et al. (2020) utilize offline RL with human feedback on sequence-to-sequence NLP tasks such as machine translation, semantic parsing, summarization and dialogue generation. Mao et al. (2020) focus on document summarization with RL, Chan et al. (2019) implement RL with adaptive rewards for neural keyphrase generation.

Furthermore, RL has been incorporated in the fine-tuning of the large language models such as ChatGPT (Christiano et al., 2023), GPT-4 (OpenAI, 2023) etc. with the methodology Reinforcement Learning from Human Feedback (RLHF) (Bradley Knox and Stone, 2008; Christiano et al., 2017; MacGlashan et al., 2017; Ziegler et al., 2019). RLHF is an approach that leverages human guidance to improve the learning process of reinforcement learning agents. RLHF incorporates human feedback, either in the form of explicit instructions or evaluative signals, to guide the agent's learning process. By incorporating human feedback, RLHF enables the generation of more coherent and contextually appropriate language by LLMs.

To conclude, the expanding interest in the amalgamation of RL and NLP underscores the potential for transformative advancements in this domain. As the fusion of these dynamic fields promises novel methodologies and solutions, our study endeavors to propose agile topologies for text classification, through comprehensive analysis and experimentation.

2. BACKGROUND

We make use of deep learning, natural language processing (NLP) and reinforcement learning (RL) in this study. We utilize deep neural network topologies such as convolutional neural networks (CNN), recurrent neural networks (RNN) especially long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997), Transformers and transformer based pretrained large language models for our purposes.

2.1 Deep Natural Language Processing

For our baseline models and the parts of our proposed model training schemes we use deep natural language processing methods, that integrate deep neural networks into NLP tasks. NLP is a subfield of computer science, artificial intelligence, and linguistics that focuses on the interactions between human language and computer programs, which has been a challenging effort for computer scientists and AI researchers (Goldberg and Hirst, 2017). Many approaches have been developed to solve the problems in NLP tasks. We will briefly mention the history of some of these approaches, and latest ones we use such as recurrent neural networks, transformers and language models.

2.1.1 Feature Extraction

In all machine learning tasks, one needs to represent the data being used numerically, even if there are some parts of the data that are not in fact given so, such as categorical variables like gender of customers or the review text of a movie criticism. Finding relevant information from data and converting this information into valuable

and meaningful numerical representations is a tedious process. This process is in general called *feature extraction* and the models that are used for this process is called *feature extractors*. In order to represent text in a meaningful and numerical way, various methods have been developed throughout the several past decades.

One straightforward approach to encode the textual data is to use one-hot encoding. Here, we can treat each individual letter or word as one category and encode the data accordingly. However, since the number of distinct words in a document can be huge, the one-hot encoding approach will introduce an extremely large dimensionality to the data which will also probably be severely sparse. This is not a feasible way to extract features and generally not used in NLP tasks.

Thus, many methods are developed to extract features from textual data in a feasible way. One of the first such methods in NLP is the Bag-of-Words (BOW) approach. Here, we look at the number of appearances of the words within the text, as a feature, this approach is also known as unigram modeling. We can also look at the number of appearances of any number of consecutive words, *aka n-gram modeling* for n consecutive words, which may be computationally dense but can extract more detailed information about the text.

We can also use quantities based on external information, for example on words that appear frequently in one document, yet appear scarcely in other documents. This way we can distinguish words that are generally common, (like stopwords *a, an, the*) from words that are associated to the document's topic. To do this, when using the bag-of-words approach, it is common to use TF-IDF (*Term Frequency-Inverse Document Frequency*) weighting (Manning et al., 2008). Consider a document d which is part of a larger corpus D. The term frequency (TF) of a word w in d is

$$\frac{\#_d(w)}{\sum_{\tilde{w}\in d}\#_d(\tilde{w})} \tag{2.1}$$

where $\#_d(w)$ is the number of appearances of word w in document d. Instead of using this as a feature for the word w, we multiply this with its inverse document frequency (IDF) where IDF of document d is defined as

$$log\frac{|D|}{|\{d \in D, w \in d\}|} \tag{2.2}$$

Thus, instead of using the count or frequency of a word, TF-IDF method uses the combined feature

$$\frac{\#_d(w)}{\sum_{\tilde{w}\in d}\#_d(\tilde{w})} \cdot log\frac{|D|}{|\{d \in D, w \in d\}|} \tag{2.3}$$

for each word w.

Another method, which we extensively make use of in our study, is to use word embeddings, which is introduced in Bengio et al. (2003) for the first time. The method of word embeddings is a way of representing words as dense vectors of real numbers, instead of representing them as one-hot vectors. These vectors capture the meaning and context of the words, so that words with similar meanings will have similar vectors, after training a machine learning model using them.

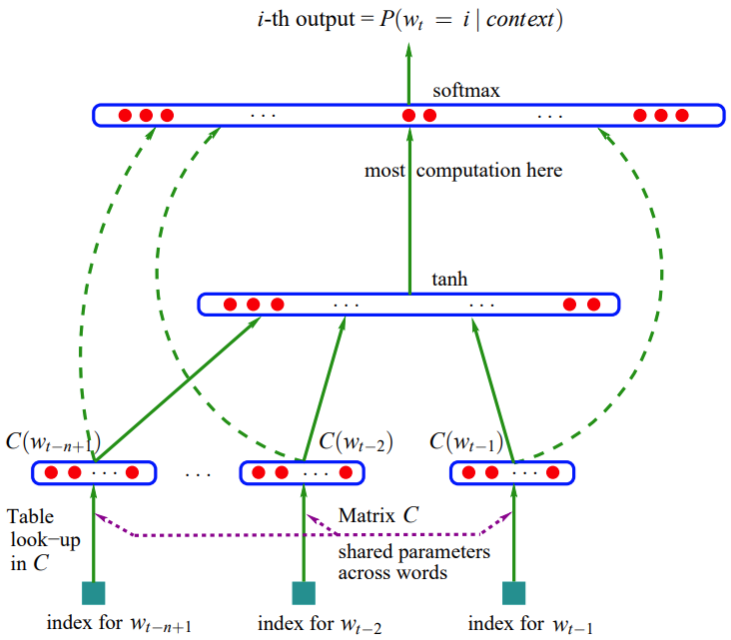


Figure 2.1 Neural network topology of the first introduced word embeddings: The shared parameters across the words are learned with backpropagation (Bengio et al., 2003).

The parameters for the mapping of words to vectors will be learned through back-propagation depending on the task of the model. This allows the model to learn relationships between words and use them to make predictions about the text. The purpose is to create a language model using neural networks with shared parameters for the mapping of words to vectors and obtain the word embeddings as a by-product. Here, neural network attempts to calculate n-gram probabilities of each word, given the context. This model and other topologies and adopted this approach induced competent word embeddings, where words with similar meanings are positioned closer together in a high dimensional vector space.

2.1.2 Neural Language Models

Language models (LMs) are models that are trained on large amounts of text data to predict the next word or a phrase in a sequence. These models are used in language generation, machine translation, and speech recognition. The goal of a language model is to assign a probability to a sequence of words, indicating how likely it is that those words would appear in a given context. This allows the model to generate text that is coherent and sounds natural to a human reader. As part of LMs, neural language models (NLMs) make use of neural networks in their models. As a by-product of NLMs we can obtain extracted features for each word from the textual data.

Neural Language Models are firstly established in Bengio et al. (2003) but were unable to be adopted extensively until the emergence of larger computational power and big data. Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), which introduced highly capable methodologies for representing words as dense and continuous vectors, incorporating the idea of word embeddings in Bengio et al. (2003), utilizing large corpora. Word2Vec model generates a language model either with Continuous Bag-of-Words (CBOW) model, where the goal is to predict an unseen word given past and future words that is present in the sentence, or Continuous Skip-Gram model, where the goal is to predict words in a certain range that come before or after given one word. On the other hand Pennington et al. (2014) use a global logbilinear regression model which blends global matrix factorization and local context window methods.

However, as NLP tasks grew in complexity, the limitations of these static word embeddings became apparent. To address these limitations, the transformer-based models emerged after the emergence of transformer topology (Vaswani et al., 2017). These models are pre-trained on large-scale corpora to learn contextualized representations. This led to the development of avant-garde models like the Generative Pre-trained Transformer (GPT) series (Radford and Narasimhan, 2018) and the Bidirectional Encoder Representations from Transformers (BERT) series (Devlin et al., 2018), which achieved state-of-the-art performance on a wide range of NLP benchmarks. These models leverage the self-attention mechanism to capture dependencies between words, enabling them to grasp both local and global context. The success of GPT and BERT variants has demonstrated the potential of pre-training language models on large amounts of unlabeled text data and fine-tuning them on specific tasks.

2.1.3 Deep Neural Network Topologies

As all topologies we use in our methodoloy are based on neural networks, we also briefly mention the history of some topologies.

2.1.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (Fukushima, 1980; LeCun et al., 1998), with their topology inspired by the visual cortex, is widely used in feature extraction especially for image analysis and understanding. The fundamental principle underlying CNNs is the application of convolution operations, which are small, learnable filters that scan through input data, extracting salient features while preserving spatial relationships. LeCun et al. (1998) introduced CNNs as a means of recognizing handwritten characters, showcasing the application of CNNs from pixels of images. Researchers also have demonstrated the effectiveness of these networks in sequence processing, particularly for tasks involving text classification and sentiment analysis. By integrating word embeddings into CNNs, these models can effectively capture contextual relationships between words (Kim, 2014).

2.1.3.2 LSTM

LSTM (Long Short-Term Memory) (Hochreiter and Schmidhuber, 1997) is a type of recurrent neural network (RNN) topology that was introduced in 1997. It is designed to address the issue of vanishing gradients in traditional RNNs, which can cause the model to struggle with long-term dependencies in sequential data. LSTM uses a gating mechanism that allows the model to selectively remember or forget information from previous time steps, making it well-suited for processing sequential data with long-term dependencies.

2.1.4 Transformer

Transformer (Vaswani et al., 2017) which is a powerful neural network architecture that was introduced in 2017. It was designed to address the limitations of traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs) in processing sequential data. Transformers are based on the self-attention mechanism, which allows the model to selectively attend to different parts of the input sequence, making it more efficient and effective at handling long-range dependencies in sequential data.

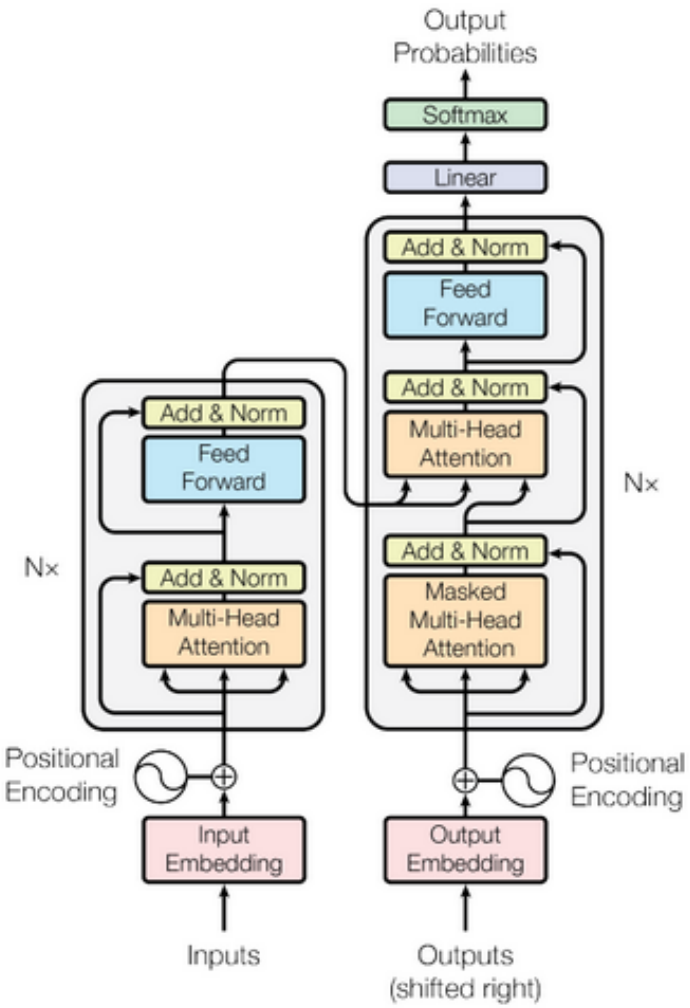


Figure 2.2 The Transformer Architecture (Vaswani et al., 2017)

The Transformer architecture consists of two main components: the encoder and the decoder. Both the encoder and decoder are composed of multiple layers, each of which contains two sub-layers: a self-attention layer and a feed-forward neural network layer.

The self-attention layer computes the attention weights between all input tokens, allowing the model to weigh the importance of each token in the context of the entire input sequence. This is done by computing a weighted sum of the values, where the weights are determined by a softmax activation function over the dot product of the query and key vectors.

The feed-forward neural network layer applies a linear transformation followed by a non-linear activation function, such as the rectified linear unit (ReLU), to the output of the self-attention layer.

Here is the mathematical equation for the self-attention mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

where Q, K, and V are the query, key, and value vectors, respectively, and d_k is the dimensionality of the key vectors.

The output of the self-attention layer is passed through a residual connection and layer normalization, before being fed into the feed-forward neural network layer. The output of the feed-forward neural network layer is again passed through a residual connection and layer normalization, before being fed into the next layer.

The decoder also includes an additional multi-head attention layer, which attends to the encoder output, allowing the decoder to generate output tokens based on the context of the entire input sequence.

2.2 Reinforcement Learning

We also incorporate some ideas from Reinforcement Learning in our proposed models hence we will give a brief summary of this field as well. Formally, Reinforcement Learning is a framework for solving control tasks by building AI, more often called agents that learn from a simulation environment by interactions through trial and error and receiving positive or negative rewards as unique feedback. The process of RL can be summarized in one diagram:

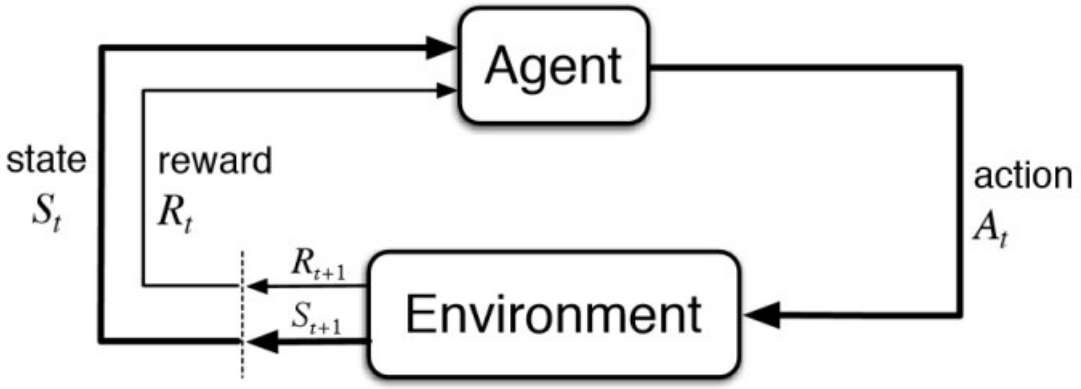


Figure 2.3 Process of Reinforcement Learning

- Initially, agent receives state S_0 from the environment.
- Based on S_0, agent takes action A_0.
- Environment changes into a new state S_1, as a result of agent's action.
- Agent receives reward R_1 from the environment.
- Agent receives state S_1 from the environment.

  ...

The agent's goal is to maximize its cumulative reward *aka* expected return, as the theory of reinforcement learning is based on the reward hypothesis (Silver et al., 2021; Sutton and Barto, 2018) in which all goals that agent might have can be described as the maximization of the expected return.

2.2.1 Definitions and Notations

Here, we give the necessary definitions and notations in the RL terminology.

Initially, the environment has a *state space*, denoted $\mathbb{S}$ which is the set of all possible states that the agent can go to.

The agent has an *action space*, denoted $\mathbb{A}$, which is the set of all possible actions that agent can take. The action space can be discrete or continuous depending on the cardinality of $\mathbb{A}$.

We assume that the environment the agent interacts with has *Markov Property* which implies that the agent needs only the current state to decide what action to

take, at any time step.

The *cumulative reward* at time step t, denoted G_t is written

$$G_t := \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}$$

where $\gamma \in [0, 1]$ is the discount rate applied to rewards and r_t is the reward obtained by the agent at time step t. We apply discount as the rewards that come sooner are more predictable than long term rewards.

The *policy* π is the function that decides for the agent which action to take, given the state it is in. In all RL problems, we aim to approximate the agent's policy to the optimal policy π^*, which maximizes the expected return if the agent acts according to it. The policy of an agent can be deterministic, i.e the policy will always return the same action at a given state:

$$a = \pi(s),\ a \in \mathbb{A}, s \in \mathbb{S}$$

or stochastic, i.e the policy will output a probability distribution over actions:

$$\pi(a|s) = P[a|s],\ a \in \mathbb{A}, s \in \mathbb{S}$$

We can train our agents mainly with two different approaches: *policy-based* approaches and *value-based* approaches. In policy-based approaches, we learn a policy function directly, where this function maps each state to the best corresponding action at that state, or a probability distribution over $\mathbb{A}$. On the other hand, in value-based approaches we train a value function denoted $\nu_\pi(s)$ that maps a state to the expected value of being at that state.

2.2.2 Value-Based Approaches

The value of a state is the expected discounted return the agent can get, if it starts at that state and act according to the policy afterwards. The *state-value function* under a policy π, with which we calculate the value of a state at time step t, is given as

$$\nu_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s],\ \forall s \in \mathbb{S}$$

The *action-value* function under a policy π, in which we calculate the value of a state-action pair S_t, A_t at time step t is given as

$$\mathbb{Q}_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.5}$$

However, one can easily see that calculating the expected return for each state-action pair can be a tedious task, especially on environments with large state spaces. To overcome this issue, we make use of *Bellman Equation*. The main idea of this equation comes from the agent's decision problem it faces at each state. We can break this decision problem into smaller subproblems with Bellman's optimality principle (Bellman, 2003) and assume that value of a current state is equal to the sum of immediate reward the agent gets coming to that state and discounted value of the following state. We can rigorously define this as

$$\nu_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma \cdot \nu_\pi(S_{t+1})|S_t = s] \tag{2.6}$$

There are mainly two different strategies on how to train a value function for value-based approaches or a policy function for policy-based approaches. In *Monte Carlo Methods*, we use an entire episode before updating the function we approximate:

$$\nu(S_t) \leftarrow \nu(S_t) + \alpha \cdot [G_t - \nu(S_t)] \tag{2.7}$$

where α is the hyperparameter for learning rate. On the other hand with *Temporal Difference Learning*, we use only a step of agent's interaction $(S_t, A_t, R_{t+1}, S_{t+1})$ to update the function. As we don't traverse through the entire episode we estimate G_t by adding the reward the agent gets with the discounted value of the next state. This estimation is called *bootstrapping*:

$$\nu(S_t) \leftarrow \nu(S_t) + \alpha \cdot [R_{t+1} + \gamma \cdot \nu(St + 1 - \nu(S_t)] \tag{2.8}$$

2.2.2.1 Q-Learning

Q-Learning (Watkins, 1989) is an *off-policy* value-based algorithm that uses Temporal Difference (TD) learning to train its agent by approximating an action-value function, called Q-Function. To summarize, this algorithm creates a table for each state-action pair called the *Q-Table* and updates the values at each time step using TD learning. Main update equation is a modified version of the equation in TD learning (2.8):

$$\underbrace{\mathbb{Q}(S_t, A_t)}_{\text{new estimation}} \leftarrow \underbrace{\mathbb{Q}(S_t, A_t)}_{\text{former estimation}} + \underbrace{\alpha \cdot [\underbrace{R_{t+1} + \gamma \cdot \max_a \mathbb{Q}(S_{t+1}, a)}_{\text{TD target}} - \mathbb{Q}(S_t, A_t)]}_{\text{TD error}} \tag{2.9}$$

where α is the hyperparameter for learning rate.

Algorithm 1: Q-Learning

Input : policy π, $num\_episodes \in \mathbb{Z}^+$, $\alpha > 0$, $\{\epsilon_i\}$ decaying rates of exploration

Output: Approximated value function $\mathbb{Q}$ ($\mathbb{Q} \approx {q_\pi}^*$ if number of episodes is sufficiently large)

Initialize $\mathbb{Q}$ arbitrarily. (e.g. $\mathbb{Q}(s, a) = 0$, $\forall s \in \mathbb{S}, \forall a \in \mathbb{A}$)
for $i \leftarrow 1$ *to* $num\_episodes$ **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    while S_t *is not terminal* **do**
        Choose action A_t using policy derived from $\mathbb{Q}$ (ϵ-greedy)
        Take action A_t, observe R_{t+1}, S_{t+1}
        $\mathbb{Q}(S_t, A_t) \leftarrow \mathbb{Q}(S_t, A_t) + \alpha \cdot (R_{t+1} + \gamma \cdot \max_a \mathbb{Q}(S_{t+1}, a) - \mathbb{Q}(S_t, A_t))$
        $t \leftarrow t + 1$
    end
end
return $\mathbb{Q}$;

As we see in pseudocode, different policies are used for acting (ϵ-greedy policy) and updating (greedy, deterministic policy), which is why Q-Learning is called an off-policy method.

2.2.2.2 Deep Q-Learning

Using a table for each state-action pair can become ineffective, even infeasible in large state space environments. In this case, one approach to overcome this issue is to approximate the Q-values of a Q-table using a parameterized Q-Function, such as a neural network.

Thus, Deep Q-Learning algorithm makes use of deep neural networks to estimate different Q-values for every possible state-action pair (Mnih et al., 2013). Differently from Q-Learning algorithm, we need to create a loss function between the Q-value predictions and the Q-target (TD target in (2.9)) and use gradient-descent to update the parameters in the neural network.

We define the Q-Target at time step t by

$$y_t := r_t + \gamma \cdot \underset{\bar{a}}{max} \widehat{\mathbb{Q}}(\phi_{t+1}, \bar{a}; \bar{\theta}) \tag{2.10}$$

where ϕ_{t+1} is the representation of the S_{t+1} calculated by the neural network with parameters $\bar{\theta}$, and $\widehat{\mathbb{Q}}$ is the target action-value function. Using this, we define Q-Loss by

$$y_t - \mathbb{Q}(\phi_t, a_t; \theta) \tag{2.11}$$

where similarly ϕ_t is the representation of the S_t calculated by the neural network with parameters θ, and $\mathbb{Q}$ is the action-value function being approximated.

We will give a slightly modified version Deep Q-Learning algorithm which provides more stability and efficiency. The algorithm has two phases:

- **Sampling**: The agent perform actions, and the observed experiences (states, rewards) are stored in replay memory.
- **Training**: Selecting a small batch of experiences randomly, agent learns from the batch using a gradient descent update.

This training process might suffer from instability as a result of combining a non-linear Q-value function approximation and bootstrapping (2.8). Therefore, we give a modified version of the Deep Q-Learning algorithm in which we apply some modifications.

Algorithm 2: Deep Q-Learning (Double DQN with Experience Replay)

Input : $C, N, T, num\_episodes \in \mathbb{Z}^+$

Output: Approximated value function $\mathbb{Q}$

Initialize replay memory D with capacity N

Initialize action-value function $\mathbb{Q}$ with random weights θ

Initialize target action-value function $\widehat{\mathbb{Q}}$ with random weights $\hat{\theta} = \theta$

for $i \leftarrow 1$ *to* $num\_episodes$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and processed sequence $\phi(s_1) = \phi_1$, where x_1 is the observation of made by the agent.

  for $t \leftarrow 1$ *to* T **do**

    With probability ϵ select a random action a_t, otherwise select $a_t = \underset{a}{argmax}\mathbb{Q}(\phi(s_t), a; \theta)$

    Execute action a_t, observe reward r_t, next observation x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and process $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1}) \in D$

    Set $y_j = \begin{cases} r_j & \text{if episode terminated at step } j+1 \\ r_j + \gamma \cdot \underset{\bar{a}}{max}\widehat{\mathbb{Q}}(\phi_{j+1}, \bar{a}; \bar{\theta}) & \text{otherwise.} \end{cases}$

    Perform a gradient descent step using $(y_j - \mathbb{Q}(\phi_j, a_j; \theta))^2$ with respect to θ (MSE Loss)

    Reset $\widehat{\mathbb{Q}} \leftarrow \mathbb{Q}$

  end

end

return $\mathbb{Q}$;

You might have noticed that we are using two different sets of neural network parameters for target action-value function and the action-value function that is being trained. This a modification that is being added to vanilla Deep Q-Learning which is called Double Deep Q-Learning (Double DQL) (van Hasselt et al., 2015), which we describe in more detail with other modification's details:

- **Experience Replay**: In online RL the agent usually interacts with the environment, get experiences, learn from the experiences and then discard them. With experience replay (Schaul et al., 2015), we create a buffer that saves experience samples (the states, actions and rewards that agent gets) that can be reused during training. This allows the agent to learn from

individual experiences multiple times, avoid forgetting previous experiences and reduce correlation between them.

- **Fixed Q-Target**: When we calculate the TD error (loss) we calculate the difference between TD target (the target policy) and the current estimation of $\mathbb{Q}$. Since we do not know what Q-target is actually is, we estimate it with Bellman equation (2.6). The problem in training neural networks with the general setting of Q-Learning (2.9) is that we use the same parameters for estimating TD-target and Q-value, resulting in a significant correlation between TD-target and parameters. This implies that at every training step, Q-values change but target values also change. To overcome this issue, we use Double DQNs.

- **Double Deep Q-Networks**: Double DQNs aka *double learning* handles the problem of overestimation of Q-values. In Q-Learning setting (2.9) we don't actually know that the action with the highest Q-value is actually the best action to take and if non-optimal actions are regularly given high Q-value estimations, the learning will become ineffective. To prevent this, two networks can be used to decouple the action selection from the target Q-value generation. Using DQN network to select the best action to take (line 7 in Deep Q-Learning) and using Target network to calculate the target Q-value of taking that action at the next state (line 12 in Deep Q-Learning) can prevent the problems we mention. Overall, Double DQN reduce the overestimation of Q-values and provides more stable and faster learning process for the agent.

2.2.3 Policy-Based Approaches

Here, we give some of the policy-based RL algorithms, in which a policy function is approximated directly. Unlike value-based methods, which strive to determine the value of each state or action, policy-based methods seek to identify the optimal policy directly without intermediating the value functions, offering a direct and often more intuitive pathway to solving complex reinforcement learning problems.

2.2.3.1 REINFORCE (Monte Carlo Policy Gradient)

REINFORCE algorithm (Williams, 1992), also known as *Monte Carlo Policy Gradient* uses an estimated return from an episode to update its parameters θ.

$$\pi_\theta(a|s) = P[a|s,\theta]$$

where $\pi_\theta(a_t|s_t)$ denoted the probability of agent selecting action a_t given the policy and state s_t.

To measure the performance of the policy, we define an objective function

$$J(\theta) := \mathbb{E}_{J\sim\pi}[R(\tau)] \tag{2.12}$$

where

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}$$

We make use of the *Policy Gradient Theorem* to find the best parameters θ and maximize the cumulative expected reward.

Theorem 2.2.1 (Policy-Gradient Theorem) *Given a policy π_θ, the gradient of the objective function defined as in* (2.12) *is*

$$\begin{aligned}\nabla_\theta J(\theta) =& \nabla_\theta \sum_{s\in\mathbb{S}} d_\pi(s) \sum_{a\in\mathbb{A}} Q_\pi(s,a)\pi_\theta(a|s) \\ \propto& \sum_{s\in\mathbb{S}} d_\pi(s) \sum_{a\in\mathbb{A}} Q_\pi(s,a)\nabla_\theta\pi_\theta(a|s)\end{aligned}$$

where $d_\pi(s)$ is the on-policy distribution of the states (the fraction of time spent in each state), $Q_\pi(s,a)$ is the action-value function as in (2.5), *$\mathbb{S}$ the state space and $\mathbb{A}$ the action space. Here, $\nabla_\theta\pi_\theta(a|s)$ gives us the direction of the steepest increase,*

using which we can change the parameters of the policy π_θ

Algorithm 3: REINFORCE (Monte Carlo Policy Gradient)

Input : A differentiable policy parameterization π_θ, step size α number of episodes N, where $\alpha, N \in \mathbb{Z}^+$, discount factor γ.

Output: Approximated policy π_θ

Initialize policy parameters θ with random weights (or to 0).
for $i \leftarrow 1$ *to* N **do**
    Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ following $\pi_\theta(\cdot|\cdot)$
    for $t \leftarrow 1$ *to* T **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\theta \leftarrow \theta + \alpha \gamma^t G \nabla log[\pi_\theta(A_t|S_t)]$
    end
end
return π_θ;

As it can be seen in the pseudocode, the parameters of the policy are updated with the equation derived from Policy-Gradient Theorem.

2.2.4 Advanced Methods

Some of the latest RL algorithms do not fall in the category of policy-based or value-based approaches, which we will give brief summaries in this section.

2.2.4.1 Advantage Actor Critic (A2C)

In REINFORCE algorithm, we calculate the return $R(\tau)$ with Monte Carlo sampling. Since the trajectories can lead to different returns due to the stochasticity of the environment, policy and the returns, we often encounter high variance. That is, starting from the same state can lead agent to obtain completely different returns.

To reduce this variance problem, Mnih et al. (2016) introduce a combination of policy based and value based methods, the Actor-Critic method. In this setting, we train two function approximators:

- A policy that controls how the agent acts: $\pi_\theta(s, a)$
- A value function to assist the policy updates by evaluating the action taken by agent: $\hat{q}_w$

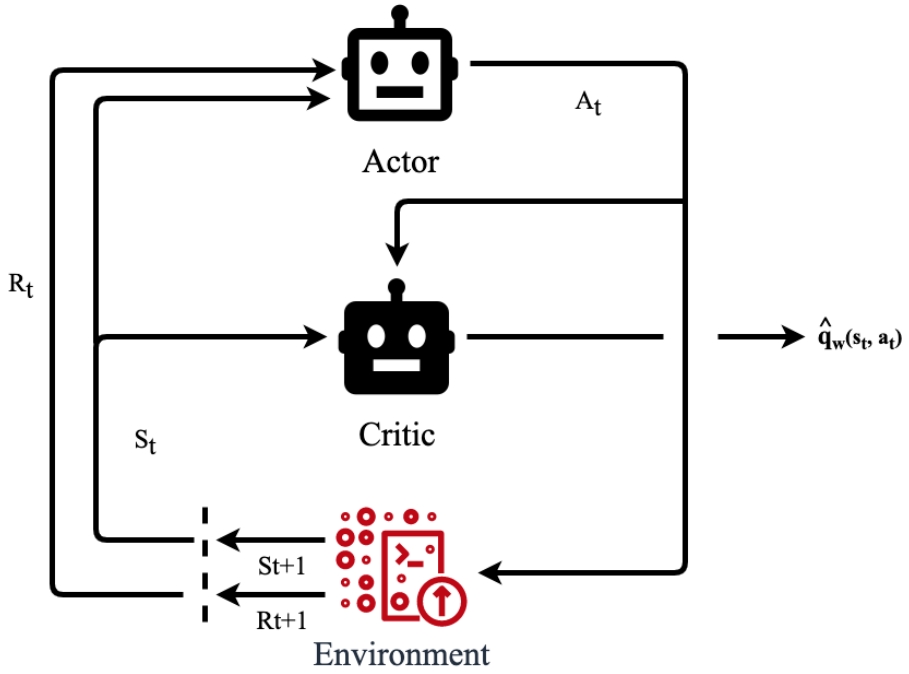


Figure 2.4 Overview of A2C algorithm

As it can be seen in figure 2.4, at each time step, both the policy π_θ (actor) and the value function $\hat{q}_w$ (critic) receives state S_t from the environment. Meanwhile, the value function also receives the action A_t that is taken by the policy after receiving S_t. The value function then computes the value of taking A_t at S_t. Also, the action A_t causes the agent (actor + critic) to go and observe a new state S_{t+1} and reward R_{t+1} from the environment.

Algorithm 4: Advantage Actor Critic (A2C)

Input : Differentiable parameterizations $\pi_\theta, \hat{q}_w$, step sizes α, β number of episodes N, where $\alpha, \beta, N \in \mathbb{Z}^+$, discount factor γ.

Output: Approximated policy π_θ, value function $\hat{q}_w$

Initialize parameters θ, w with random weights (or to 0).
for $i \leftarrow 1$ *to* N **do**
  $t \leftarrow 1$
  Initialize S_t (first state of the episode)
  $S \leftarrow S_t$
  while S *is not terminal* **do**
    $A_t \sim \pi_\theta(\cdot|S)$
    Actor takes action A_t, observe new state S_{t+1} and reward R_{t+1}
    Critic takes A_t and S_t and computes $\hat{q}_w(S_t, A_t)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta [log \pi_\theta(S_t, A_t)] \hat{q}_w(S_t, A_t)$ (update parameters of actor)
    With updated parameters, actor takes A_{t+1} given S_{t+1}
    $w \leftarrow w + \beta[R_{t+1} + \gamma^t \hat{q}_w(S_{t+1}, A_{t+1}) - \hat{q}_w(S_t, A_t)] \cdot \nabla_w \hat{q}_w(S_t, A_t)$ (update parameters of critic)
    $t \leftarrow t + 1$
    $S \leftarrow S_t$
  end
end
return $\pi_\theta, \hat{q}_w$;

2.2.4.2 Proximal Policy Optimization (PPO)

In Proximal Policy Optimization (PPO)(Schulman et al., 2017), the policy network is updated using a clipped surrogate objective function, which helps to prevent drastic policy updates that can lead to instability. The objective is to maximize the expected cumulative reward as it is in all RL tasks, while ensuring that the policy update remains within a specified range, by constraining the policy update. Clipped surrogate objective is defined as

$$L^{clip}(\theta) := \mathbb{E}_t[min(r_t(\theta)\hat{A}_t,\ clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.13}$$

where

$$r_t(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)}$$

is the fraction that measures the difference between old and updated policy parameters given that A_t the action taken by and S_t the state observed the agent at time step t. With *clip* function, maximum amount of deviation that can be made in each parameter updated can be controlled:

$$clip(r_t(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, \text{ if } r_t(\theta) < 1-\epsilon \\ r_t(\theta) \text{ if } 1-\epsilon \leq r_t(\theta) \leq 1+\epsilon \\ 1+\epsilon \text{ if } 1+\epsilon < r_t(\theta) \end{cases}$$

In the PPO algorithm in each iteration, a batch of trajectories is collected by executing the current policy in the environment, consisting of sequences of states, actions, rewards, and next states. These collected trajectories are used to compute advantages $\hat{A}_t$ in each time step t, estimating the relative value of taking a specific action compared to the average action value in that state. Advantages provide a measure of how much better or worse an action is compared to the average action, taking into account the expected cumulative reward. We give details on a later variant called Generalized Advantage Estimation (GAE) developed in (Schulman et al., 2018), incorporating both immediate and future rewards, GAE provides a more accurate estimation of the advantages and helps to improve the learning efficiency of RL algorithms. Given $\delta_t^V := r_t + \gamma V(S_{t+1}) - V(S_t)$, GAE is defined as

$$\hat{A}_t = \hat{A}^{GAE(\gamma,\lambda)} := \sum_{l=0}^{\infty} (\lambda\gamma)^l \delta_{t+l}^V \tag{2.14}$$

where γ, λ are hyperparameters for discount and bias-variance tradeoff and V is the value function estimate of the *critic* part of the algorithm.

The value network is trained to estimate the expected cumulative reward, which is used to compute advantages. By minimizing the mean squared error (MSE) loss between the predicted values and the actual rewards, the value network learns to approximate the expected cumulative reward accurately.

The policy network is updated using the clipped surrogate objective which aims to maximize the probability of taking actions that yield higher advantages while simultaneously constraining the policy update within a specified range. This prevents large policy updates that could result in a large deviation from the previous policy and lead to instability. Thus, PPO ensures that the new policy is not too different from the old policy, providing a more stable learning process.

Algorithm 5: Proximal Policy Optimization

Initialize policy network π_θ and value network V_ϕ with random weights
Initialize hyperparameters $\lambda, \theta, \epsilon \in [0, 1]$
repeat
    for *actor* $\leftarrow 1$ *to* N **do**
        Run policy $\pi_{\theta_{old}}$ old in environment for T time steps
        Compute advantages $\hat{A}_1, ..., \hat{A}_T$ using the generalized advantage estimation (GAE) with (2.14)
    end
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$ using (2.13)
    $\theta_{old} \leftarrow \theta$
until *convergence*;

3. DATASETS

There are two datasets that are used as benchmarks for text classification this study that are also used in (Yu et al., 2018) which is our main inspiration.

Dataset	# Classes	# Training/Validation Samples	# Test Samples
IMDB Movie Reviews	2	30000	25000
AG News	4	120000	9600

Table 3.1 Dataset Statistics

3.1 IMDB Movie Reviews Dataset

For sentiment analysis, we use the large movie review dataset (Maas et al., 2011) which is an extensive corpus designed for binary sentiment classification, featuring a significantly larger volume of data compared to previous benchmark datasets (Pang and Lee, 2005). The dataset consists of 25,000 highly polar movie reviews allocated for training, and an equal number for testing.

3.1.1 Examples from IMDB reviews

Review 1: "First off let me say, If you haven't enjoyed a Van Damme movie since bloodsport, you probably will not like this movie. Most of these movies may not

have the best plots or best actors but I enjoy these kinds of movies for what they are. This movie is much better than any of the movies the other action guys (Segal and Dolph) have thought about putting out the past few years. Van Damme is good in the movie, the movie is only worth watching to Van Damme fans. It is not as good as Wake of Death (which i highly recommend to anyone of likes Van Damme) or In hell but, in my opinion it's worth watching. It has the same type of feel to it as Nowhere to Run. Good fun stuff!" (**Label**: *Negative*)

Review 2: "The story centers around Barry McKenzie who must go to England if he wishes to claim his inheritance. Being about the grossest Aussie shearer ever to set foot outside this great Nation of ours there is something of a culture clash and much fun and games ensue. The songs of Barry McKenzie(Barry Crocker) are highlights." (**Label**: *Positive*)

3.2 AG News Dataset

AG News dataset (Zhang et al., 2015) is a compendium comprising over one million news articles sourced from in excess of 2000 news outlets within a period of one year, as compiled by ComeToMyHead which is an academic news search engine that has been operational since July 2004. The dataset is made available by the academic community exclusively for research in data mining (clustering, classification, etc.), information retrieval (ranking, search, etc.), XML, data compression, data streaming, and all other non-commercial undertakings.

3.2.1 Examples from AG News

News Article 1: Sony said Thursday that it would begin selling eight new television models outfitted with semiconductors designed to produce sharper images, including two with a lighting system that the company says is the world's first. (**Category**: *Science & Technology*)

News Article 2: "The president's plan to partially privatize Social Security

probably won't adversely impact financial markets, even if the program entails hundreds of billions of dollars in debt", Treasury Secretary John W. Snow said Friday. (**Category**: *Business*)

News Article 3: Liverpool were left stunned as Lomana LuaLua grabbed a dramatic stoppage-time equaliser to earn Portsmouth a 1-1 draw at Anfield. Goalkeeper Jerzy Dudek, back in the side with Chris Kirkland injured, looked. (**Category**: *Sports*)

News Article 4: Passengers face more delays, British Airways has cancelled four domestic flights as the effects of yesterday's problems continued to disrupt operations from Heathrow. (**Category**: *World*)

4. Methodology

Here, we describe our methodology of training RL policies for text classification tasks. As we discussed previously in Section 1, our objective is to create agile and lightweight topologies that can make inference and be trained rapidly. We train the policies with REINFORCE (Monte Carlo Policy Gradient) algorithm as explained in Section 2.2.3.1 using the components we designed. The components of our training scheme are the environment that the policies interacts with, the action spaces of the policies which it can select an action based on each observation it gets, the reward functions which are used as feedback signals in the parameter updates of the agent, the feature extractors that the agent uses to create a numerical representation of the observations it makes and episodic memory component allows the agent to keep track the previously made observations and actions taken.

4.1 Environment

For each text classification dataset $\mathcal{D} = \{(s_i, l_i)\}_{i=1}^{n}$ where (s_i, l_i) is a text-label pair, we create an environment that agent will interact. Given a text sample from the dataset, we create an observation for the agent as follows:

- Tokenize and numerically encode the textual input using either a manual tokenizer or a pretrained tokenizer. Pad or truncate the obtained sequences so that each sequence has the same length M.
- Select chunk length m and split the tokenized vectors into chunks of this length. For the last chunk, we pad the vector into the suitable size with zeros. (m is an hyperparameter that can be tuned).

Applying these for each text sample we obtain a new set as $\mathcal{E} = \{c_1, c_2, ..., c_n\}$ where n is the number of samples in the data and $c_i = [c_{i_1}, c_{i_2}, ..., c_{i_m}]$ the chunks in the

text sample. It should also be noted that for each $i = 1, 2, ..., n, j = 1, 2, ..., m$ we have $c_{i_j} = w_{i_{j1}}, w_{i_{j2}}, ..., w_{i_{jk}}$, the words/tokens in the chunk where $k = \frac{M}{m}$ is the number of tokens in each chunk. At each time step, the agent observes extracted features from feature extractor which takes c_{i_j} as an input, and takes an action from the action space.

4.2 Overview of the Policies

We define three policies to solve the text classification task efficiently by deciding to stop reading, reread or skip some parts of the text. We define *stop policy* that chooses the stop or continue reading, *classifier policy* and *next policy*.
At each time step, *stop policy* decides to stop or continue reading. If stop action is taken *classifier policy* predicts a label , otherwise *next policy* chooses the number of chunks that will be skipped. All policies observe the extracted features from the *feature extractor* in this scheme.

Initially, the topology observes text chunk at position n with feature extractor. *Stop policy* decides the action *continue* , prompting the *next policy* which decides to skip k chunks. Then feature extractor observes the text chunk at position $n + k$, causing the *stop policy* to take action *stop*. Prompted by this, *classifier policy* makes a prediction of the observed texts so far. Note that all policies use the outputs of the feature extractor using the outputs of the feature extractor as input and meanwhile, hidden and cell states of the LSTM are kept in memory until the end of the episode.

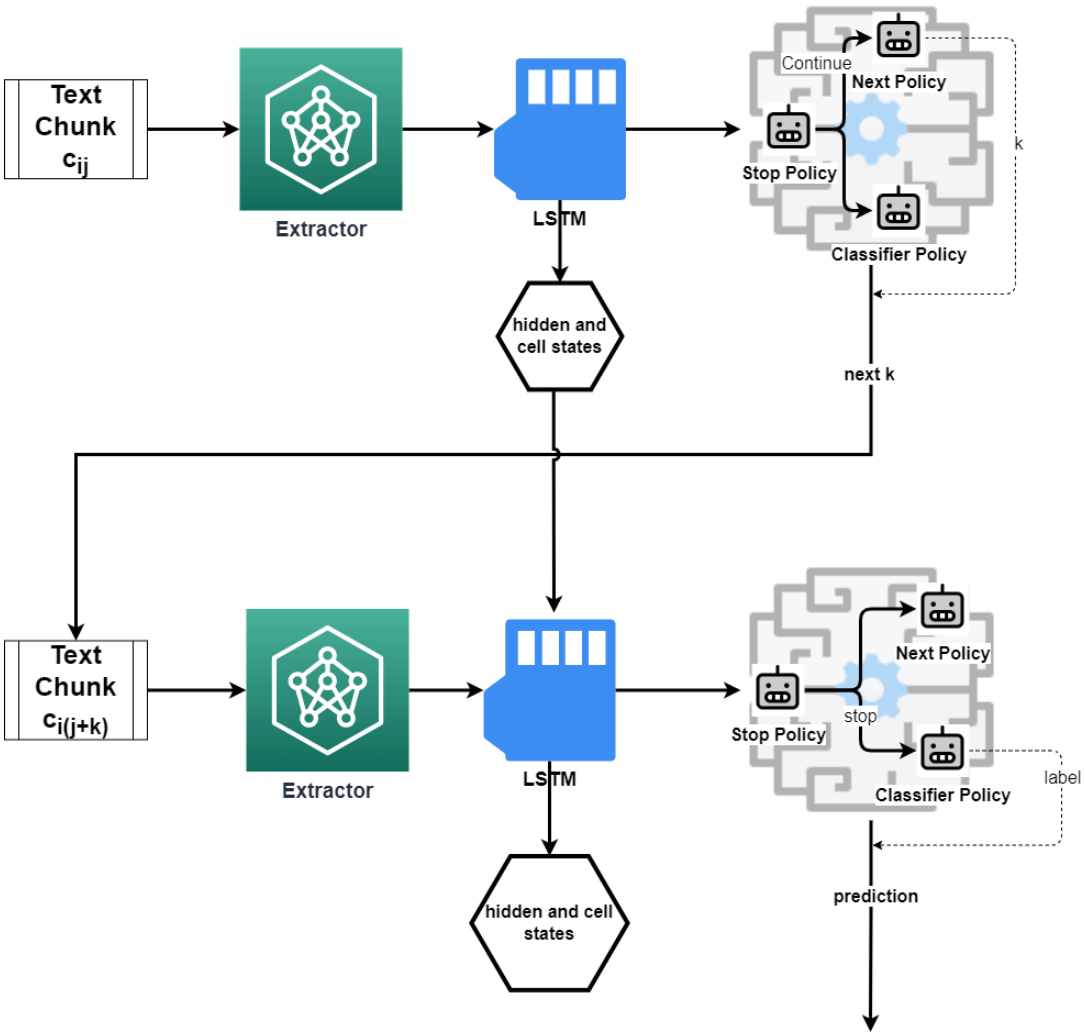


Figure 4.1 Overview of one episode

4.3 Action Spaces

We define three action spaces for three policies we use in our training scheme. We define $\mathcal{A}_{\mathcal{S}} = \{stop, continue\}$ the action space for the *stop policy*.
Given the data $\mathcal{D} = \{(s_i, l_i)\}_{i=1}^{n}$, we know that each s_i has a unique label $l_i \in \{label_1, ..., label_p\}$ where p is the number of possible labels. Hence, we define the action space for the *classifier policy* as $\mathcal{A}_{\mathcal{C}} = \{label_1, ..., label_p\}$.

Lastly, we define $\mathcal{A}_{\mathcal{N}} = \{next_0, next_1, ..., next_k\}$ for the *next policy*.

4.4 Reward Functions

We make use of three reward functions as discussed in Yu et al. (2018), Yu et al. (2017), Ramamurthy et al. (2023). After each action of the *stop policy*, an action is taken either by *classifier policy* or *next policy*. Thus, the rewards at time step t given action $a_t \in \mathcal{A}_\mathcal{C} \cup \mathcal{A}_\mathcal{N}$ and correct label l_t is calculated as follows:

$$R_{1_t} := \begin{cases} -\mathcal{L}(y, \hat{y}) - \alpha \mathcal{F}_t, \text{ if } t \text{ is the final step} \\ -\alpha \mathcal{F}_t \text{ otherwise} \end{cases}$$

where $\mathcal{L}$ is the cross-entropy loss between predicted and real label and $\mathcal{F}_t$ is the FLOP count for that step, normalized with α hyperparameter (Yu et al., 2018). We also define

$$R_{2_t} := \begin{cases} +1, \text{ if a correct prediction is made} \\ -1, \text{ if an incorrect prediction is made} \\ 0, \text{ otherwise} \end{cases}$$

as in Yu et al. (2017).

4.5 Feature Extractors

We make use of several different deep neural network topologies in order to extract features from the observations.

We use convolutional neural networks in our effort to reproduce the results in Yu et al. (2018) so as to use it as a baseline. As the details are not given, we choose embedding size as 100 for the embedding layer and do not use pretrained word embeddings. We feed the output of the embedding layer to a convolutional layer with 128 filters and kernel size 5.

For our novel topologies, we replace the convolutional neural networks with transformers and transformer-based language models. We implemented vanilla transformer with various hyperparameters and several language models for the feature extractor. However, we will only give details on the two best performing ones, which

are DistilBERT (Sanh et al., 2020) and RoBERTa (Liu et al., 2019).

DistilBERT is one of the finest language models introduced in 2019. It is a smaller and faster version of the popular BERT model, achieved by removing certain layers and using distillation technique to transfer the knowledge from the larger model to the smaller one. It has accomplished similar performance to BERT on a range of NLP tasks while requiring less computation and memory, making it a favourable choice for our application where efficiency is a priority and we have limited computational power.

On the other hand, RoBERTa (Robustly Optimized BERT Pretraining Approach) is a variant of the BERT that modifies key hyperparameters in BERT, such as removing the next-sentence pretraining objective, and training with larger mini-batches and different learning rate schedules. It is also trained on an extended version of the BookCorpus dataset (Zhu et al., 2015). These modifications enabled RoBERTa to achieve competitive performance on various NLP benchmarks, hence we extensively employ this model in the extractor as well.

4.6 Episodic Memory Component

We make use of LSTM for the episodic memory component of our topology which is designed to empower the agent to effectively retain and recall text chunks that have been previously encountered during its traversal. The LSTM is structured to accept outputs from feature extractors, together with a hidden state and the a cell state. These states are initiated as zero-tensors in the beginning of each episode and are updated and are forwarded to the LSTM again until the end of an episode within each forward pass. The outputs of LSTM are fed as inputs to the policy networks that produce the actions in the RL scheme. When a classification is made, the states are reinitialized as zero-tensors as it indicates the end of the episode.

However, in ablation studies we made where the LSTM was not utilized as the episodic memory component, we detect that topology displayed difficulty in two areas: first, in generating accurate classifications using state representations from the feature extractors; and second, in establishing a cohesive policy for navigating the text throughout episodes. The primary issue was the model's tendency towards making seemingly random decisions on all policies in the absence of an episodic memory component. This propensity towards randomness suggests the overall topology's

inability to identify and learn from patterns, emphasizing the pivotal role of the LSTM as episodic memory.

4.7 Policy Models

For all policy models, we make use of fully-connected neural networks with three hidden layers of 64 units, with input dimension the same as the output dimension of the extractor, and the output dimension as the length of action space for the corresponding policy. For each hidden layer we use ReLU activation function except the output layer, in which we use Softmax activation. We choose these hyperparameters exactly same as in Yu et al. (2018).

5. Results and Discussion

In this chapter we share our results and discuss the performance of our proposed scheme. As the tasks we use our architecture to solve is classification, we use the metrics *accuracy, precision, recall and F1 score*, commonly used in classification tasks, to measure and benchmark the performance of our proposed scheme. First, we examine the behaviours learned by the policies with different topologies we implemented.

5.1 Policy Behaviours

5.1.1 Policy Behaviours on Reward Functions on Different Data

We observe mainly three different behaviours on policies when trained on the defined rewards. We see these behaviours on both reward functions R_1 and R_2 and found little meaningful correlation between behaviour change and using R_1 or R_2.

Firstly, we observe that the agent only reads the first chunk to make a classification without reading the remaining part of the samples. We see this behaviour mostly on AG News dataset, in which we observe that it is relatively simpler to make a prediction with the initial part of the news articles. This behaviour is also seen if we increase the α parameter that penalizes the reading other chunks in R_1 too much as well but only on IMDB data. We refer this behaviour as *First-Chunk Focus*.

Secondly, we observe that the agent reads as much as it can before making a prediction, regardless of the FLOP penalty given in the reward function R_1. We see this behaviour mostly on IMDB data where we observed that the reviews are more

difficult to classify and users can take their time before they write sentences relevant to their sentiments in the reviews. We name this behaviour as *Full-Read Tendency*.

Lastly, we observe that the agent always decides to skip two chunks four times before making a classification. This is the best behaviour and closest one as attributed in Yu et al. (2018), which we think that is a local optimum policy difficult to encounter or reproduce. This behaviour is encountered with R_1 reward which we interpret as a way to optimize FLOP penalty and classification performance. We will use the mnemonic *Skip-Twice Strategy* for this behaviour.

5.1.1.1 Policy Decisions on Sample Reviews with *Skip-Twice Strategy*

Now, we give some examples on how the policies behave on sample reviews with *Skip-Twice Strategy*. To make it easier to follow, we give the parts of text observed by the agent in bold.

Review 1: *"**I am still shuddering at the thought of EVER seeing this movie again have seen action films have** even liked quite few of them but this one goes over the top. Not only does it have the **worst male actor ever Sly Stallone playing the lead role but the plot of the movie is so stupid** from the beginning why not rob the money while the plane is on the ground would be hell of lot **easier that it requires person with IQ less than his shoe number to believe it. Furthermore the plot has** no real twists at all three year old kid could guess what comes next It is set of cliches of action genre with Sly performing even worse than his other movies he was better even in Rambo III if you watch that movie as comedy rather than action film Now there is an actor who can act surprised sad anything **else than his basic face would still like to point out that this movie has two factors that** might make some people like it EXPLOSIONS are outstanding but then you can see better on the th of July LANDSCAPES are magnificient but then there are documentaries about the Alps and Himalayas so you can see better sights that way rather than waste time on this flick. Go watch some other movie instead there are hundreds even thousands better action movies."*

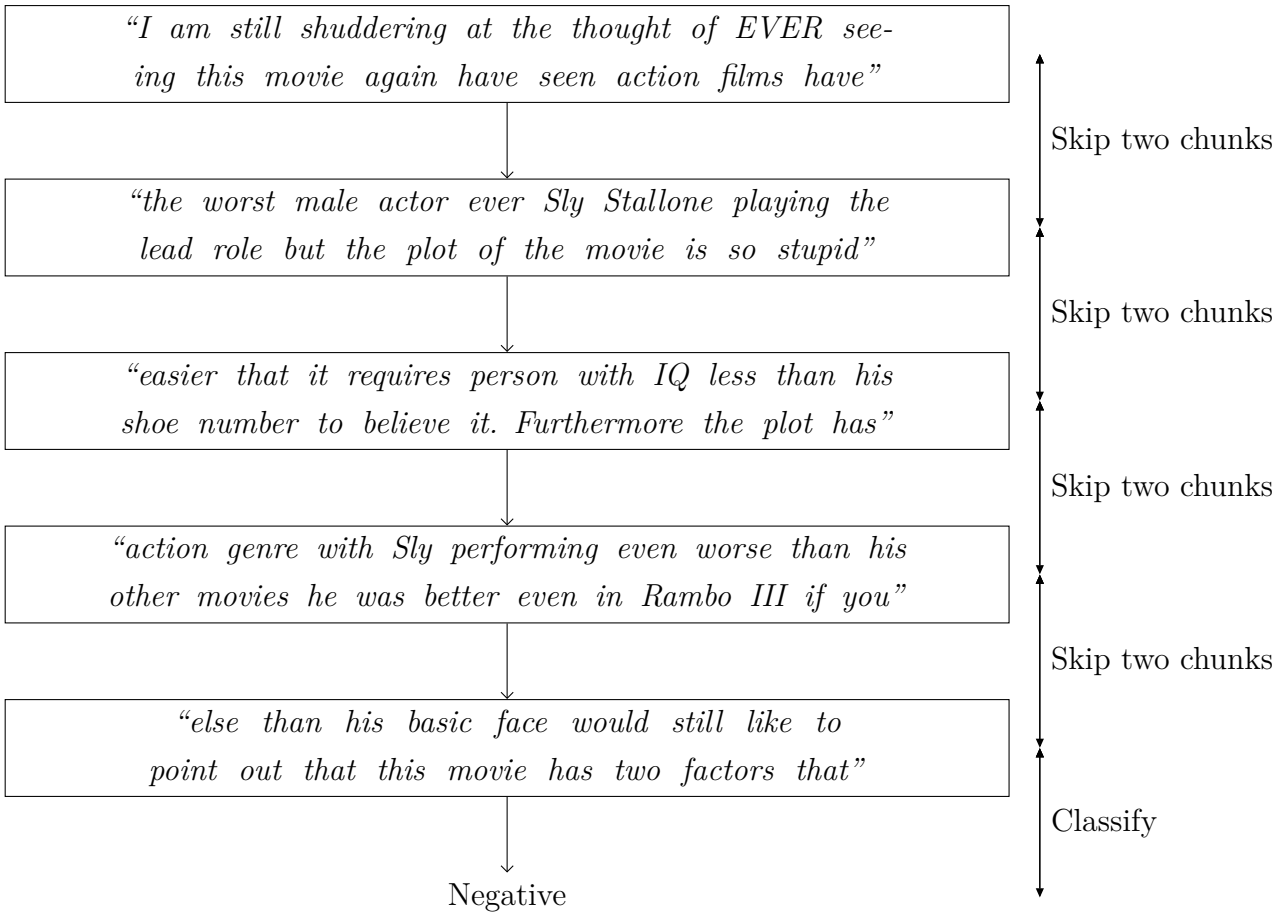


Figure 5.1 Policy correctly predicts *Negative* by skimming and stopping reading early

Review 2: *"**Within the realm of Science Fiction two particular themes consistently elicit interest were initially explored in the literature of pre** cinematic era and have since been periodically revisited by filmmakers and writers alike with varying degrees of success The first **theme that of time travel has held an unwavering fascination for fans of film as well as the written word** most recently on the screen with yet another version of the G Wells classic The Time Machine The second theme **which also manages to hold audiences in thrall is that of invisibility which sparks the imagination with it seemingly endless** and myriad possibilities And this theme too has again become the basis for film adapted from another G Wells classic **The Invisible Man the realization of which here is Hollow Man directed by Paul Verhoeven and starring Kevin Bacon and** Elisabeth Shue Sebastian Caine Bacon and his colleagues have for some time been conducting experiments for the S **Government exploring the possibility and practicality of invisibility which they have at last achieved in number of the primates upon** which they have tested their method They have in fact progressed to the point that effecting the invisibility is assured their*

only problem now is bringing the subject back to the original visual state of being It a problem however that Caine after diligent effort and too many hours in the lab has solved or so he thinks And when the application of his theory on live subject is successful he decides to present the results to the board of directors in an effort to thereby maintain the funding necessary for the continuation of the project. At the last minute though Caine demurs fearing that control of the project will be wrested from him before they can proceed to the next level the testing of human subject And he takes it upon himself to become that subject securing the assistance of his research team by telling them that they ve been given approval by the board to do so But something goes wrong and Caine becomes trapped in his cloak of invisibility and as he and his team struggle to find the solution to his considerable dilemma before it too late it all begins to take toll on Caine mind And suddenly his fear of losing funding and control becomes inconsequential as he finds himself facing the imminent" (...remaining part truncated after 400 words)

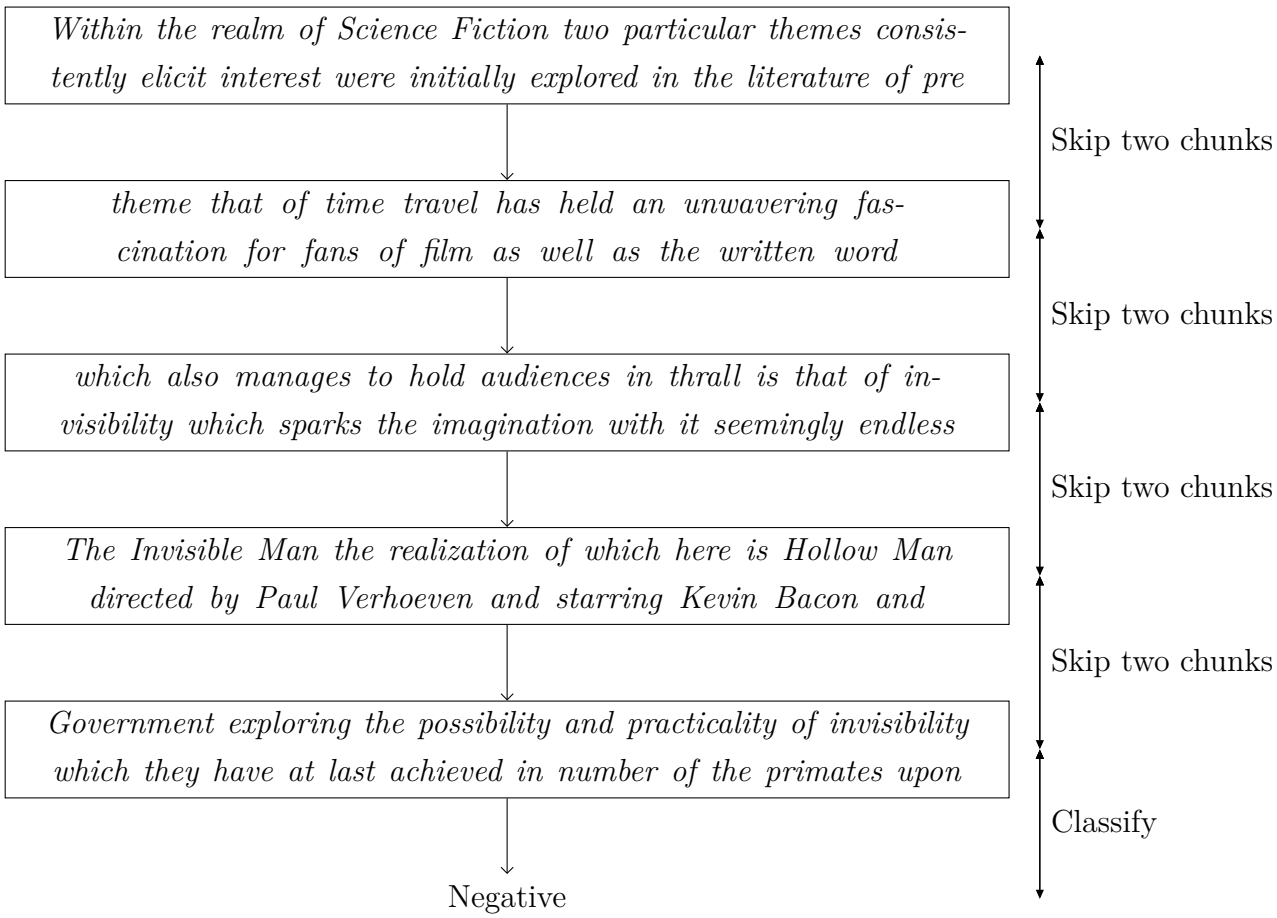


Figure 5.2 Policy correctly predicts *Negative* by skimming and stopping reading early

Review 3: ***Can Scarcely Imagine Better Movie Than This. Hey before you all go Chick Flick on me am very*** *Large Strong Masculine Macho Man who happens to think this was one of the better movies of the last years.* ***The acting was Superb and the Story was Marvelous This is wonderful medicine for the heart and soul*** *The Acting could not have been better nor the movie better cast have known for Good while that* ***Mercedes Ruehl along with Holly Hunter Joan Plowright Dame Edith Evans Sissy Spacek Judi Dench is among the greatest actresses*** *ever to appear on film And of course Cloris Leachman also in this film in my view may in fact* ***exceed them all in the shear magnum of her talent and varied roles she has appeared in over the years*** *At any rate this was an Amazing cast This film was like book that you cannot lay down and when* ***you have reached the last page wish for more still more cannot for the life of me understand why this*** *film here on the IMDb only rates That rating here is utterly Amazing to me Or perhaps not Perhaps in fact do understand it ever so well and that is what makes me really sad It makes me ever so sad that films like American Beauty Leaving Las Vegas Sexy Beast and Fight Club ratings skyrocket off the charts in popularity when they in fact at least in this viewers opinion should have received an rating that is for Rubbish Hey k realize there are lot of different stories in this world for lot of different audiences but it is sad commentary when this lovely powerful extraordinarily Directed Acted and written film seems to be over looked It obviously was at the Academy Awards as well How Sad And How predictable My summation is that if you want to see powerful Happy Sad beautiful story watch preferably own this film (...remaining part truncated after 400 words)*

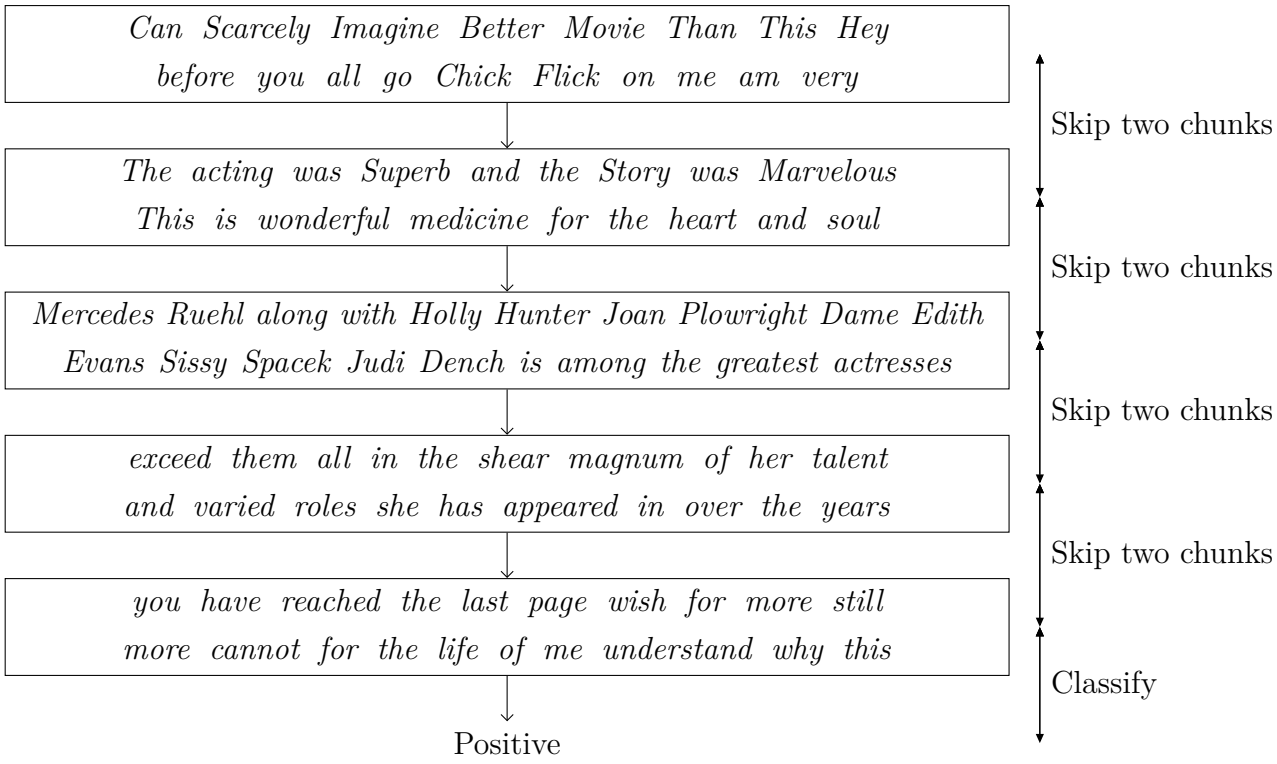


Figure 5.3 Policy correctly predicts *Positive* by skimming and stopping reading early

Upon examining the action statistics associated with the *Skip-Twice Strategy*, it becomes evident that the policies consistently opt to bypass two chunks on four occasions before proceeding with classification, showcasing this behavior at any moment after only a few training iterations. This indicates that the policies don't genuinely discern which portions of the text are salient for the classification task. Instead, they select a baseline quantity of chunks to use, in our case specifically five chunks or approximately 100 words, and then execute the classification based on this confined selection. We depict this as a result of a convergence to a rare local optimality which enables to topology to maximize the expected reward by reading only five chunks, avoiding the penalty for the reading more text in R_1 whilst making enough observations to make correct predictions.

5.2 Baseline Model Performance

We compare our results obtained with Language Model feature extractor topologies on the main benchmark with CNN feature extractor implemented by Yu et al. (2018).

On all these topologies, LSTM is used as episodic memory component for policies. Despite employing certain techniques from closely-related study Yu et al. (2017) to compensate for insufficient information provided in the paper, we were unable to attain the high levels of accuracy and balance between accuracy and efficiency purportedly achieved by the proposed model. Number of training steps, dimensions of the LSTM layers, dimensions of the word embeddings, usage of pretrained word embeddings are several such details that are not given in Yu et al. (2018), and no implemented code is provided by them either. Furthermore, while the model was reported to have superior performance when employing Advantage Actor-Critic, the available information regarding this methodology is limited.

Regardless, we implemented the closest possible topology using the same hyperparameters given in (Yu et al., 2018), and the results we observe for this scheme shows that the policy usually converges to *Full-Read Tendency* when we use reward function R_2 and to *First-Chunk Focus* when we use reward function R_1. On same rare cases, we also observe *Skip-Twice Strategy* when reward function R_2 is used, and this produced the best result on the test set of IMDB.

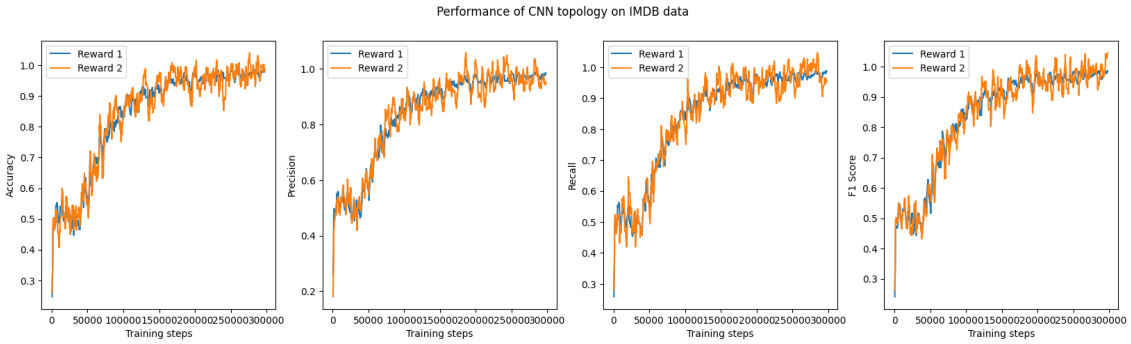


Figure 5.4 CNN Topology's training performance on IMDB dataset (left to right: Accuracy, Precision, Recall, F1 Score)

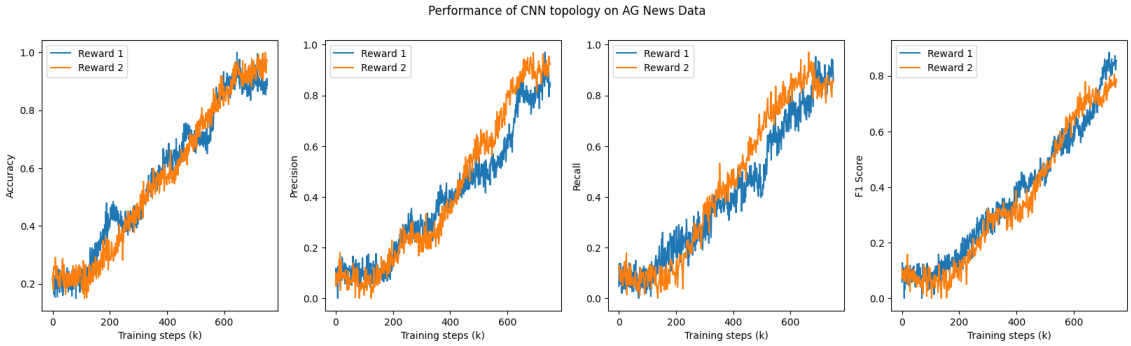
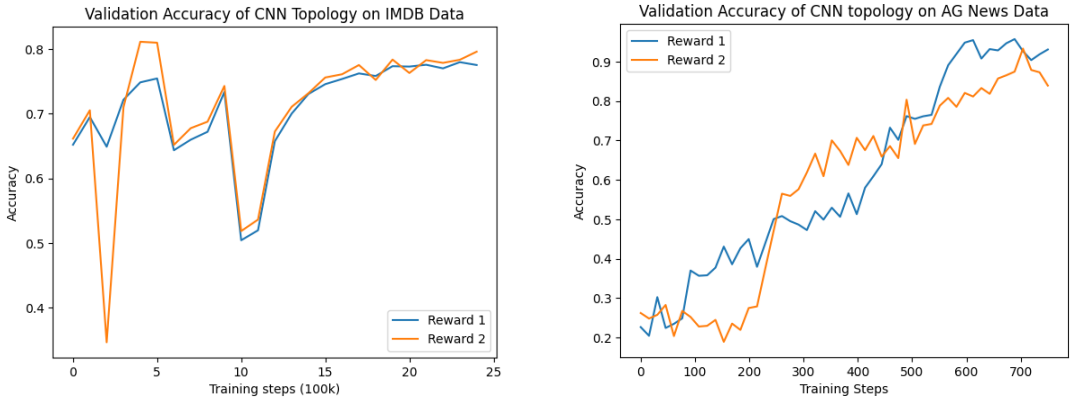


Figure 5.5 CNN Topology's training performance on AG News dataset (left to right: Accuracy, Precision, Recall, F1 Score)

On training data we see a consistent increase on all metrics. However, the perfor-

mance of the topology is not as high in the test set, implying some overfitting on the training data despite the counter-measures we applied.



(a) Performance on IMDB (b) Performance on AG News

Figure 5.6 CNN + LSTM topology performance on two datasets' validation split

The topology achieves 0.76 ± 0.008 accuracy on the test set of IMDB and 0.85 ± 0.013 accuracy on the test set of AG News when trained with the two reward functions on different seeds.

5.3 Performances of our Implemented Topologies

We use transformers and transformer based several language models as feature extractors in our experiments, where DistilBERT (Sanh et al., 2020) and RoBERTa (Liu et al., 2019) were the most successful ones.

5.3.1 DistilBERT + LSTM

We were able to achieve performant policies when we made use of DistilBERT as the feature extractor. Since it is a lightweight version of the BERT, we could test this topology more thoroughly with the limited resources we had. We observe that the policies trained with this topology mostly converge to *First-Chunk Focus*. We depict this as a result of generality and robustness of DistilBERT, making it unnecessary for the topology to read more text to successfully make predictions. Usage of reward

functions R_1 and R_2 does not seem to make difference on the behaviours on AG News, albeit we see *Skip-Twice Strategy* when we use R_1 and *Full-Read Tendency* when R_2 is used on IMDB data on some occasions. One can speculate that the penalty applied for reading causes the policy to this behaviour change.

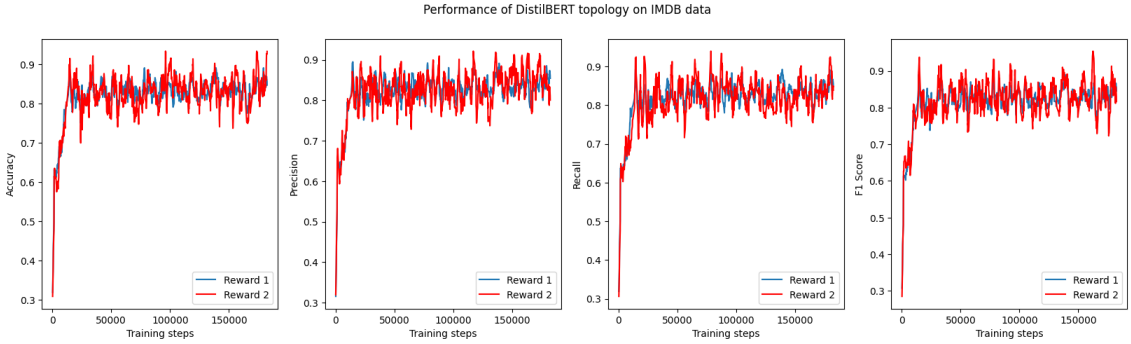


Figure 5.7 DistilBERT Topology's training performance on IMDB dataset (left to right: Accuracy, Precision, Recall, F1 Score)

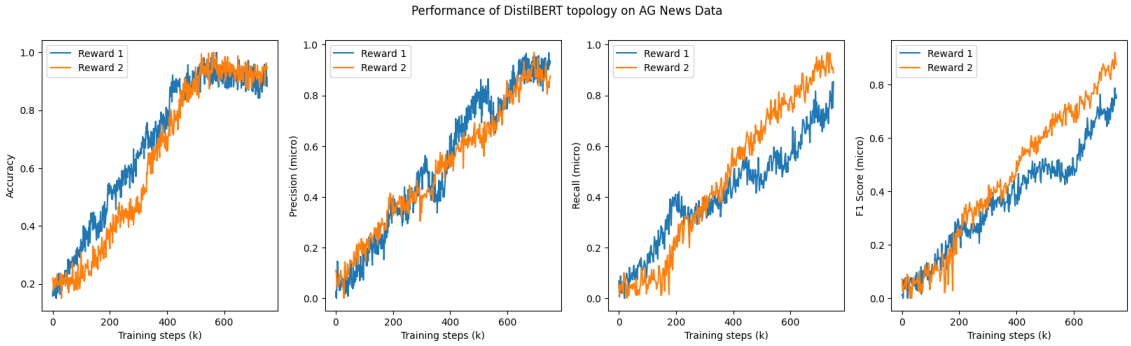
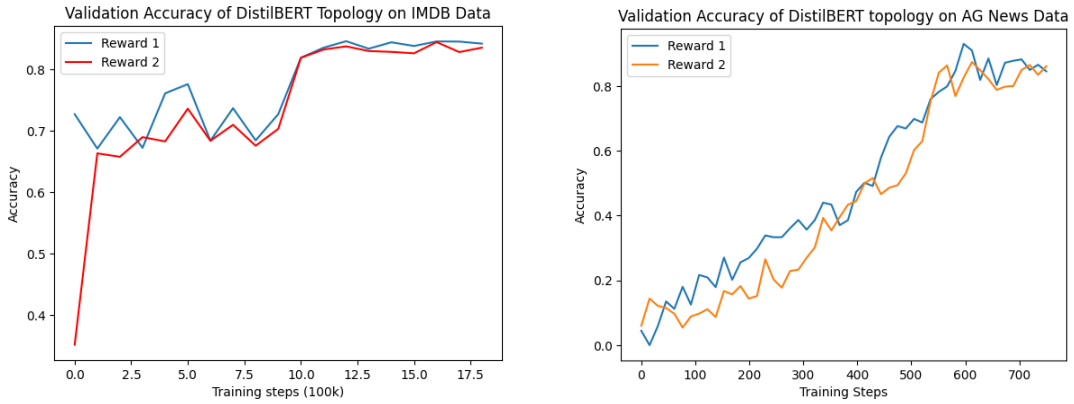


Figure 5.8 DistilBERT Topology's training performance on AG News dataset (left to right: Accuracy, Precision, Recall, F1 Score)



(a) Performance on IMDB

(b) Performance on AG News

Figure 5.9 DistilBERT + LSTM topology performance on two datasets' validation split

This topology achieves 0.81 ± 0.011 accuracy on the test set of IMDB and 0.91 ± 0.007 accuracy on the test set of AG News.

5.3.2 RoBERTa + LSTM

Another language model with which we accomplish propitious performance is RoBERTa, when used as the feature extractor. We perceive that the policies in this topology converge either to *First-Chunk Focus* or *Full-Read Tendency* independent from the used reward function. We encounter *First-Chunk Focus* on AG News, for which it can be argued that it is easier to make correct predictions with less text, and *Full-Read Tendency* on IMDB which is arguably more difficult to classify.

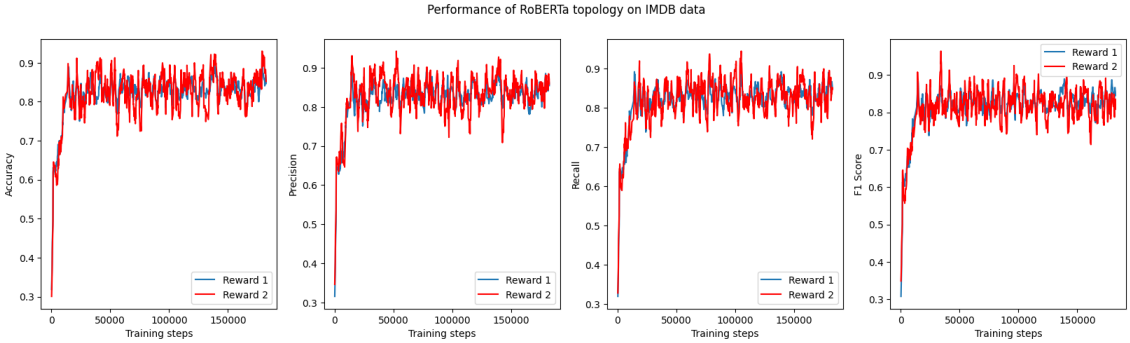


Figure 5.10 RoBERTa Topology's training performance on IMDB dataset (left to right: Accuracy, Precision, Recall, F1 Score)

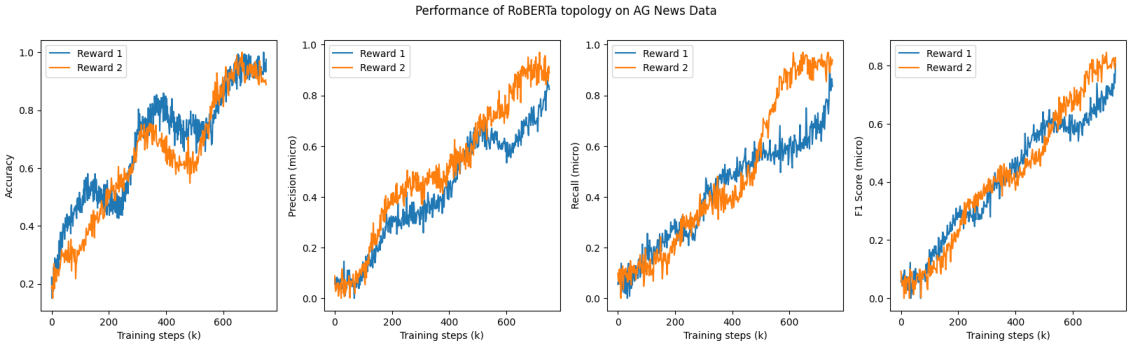
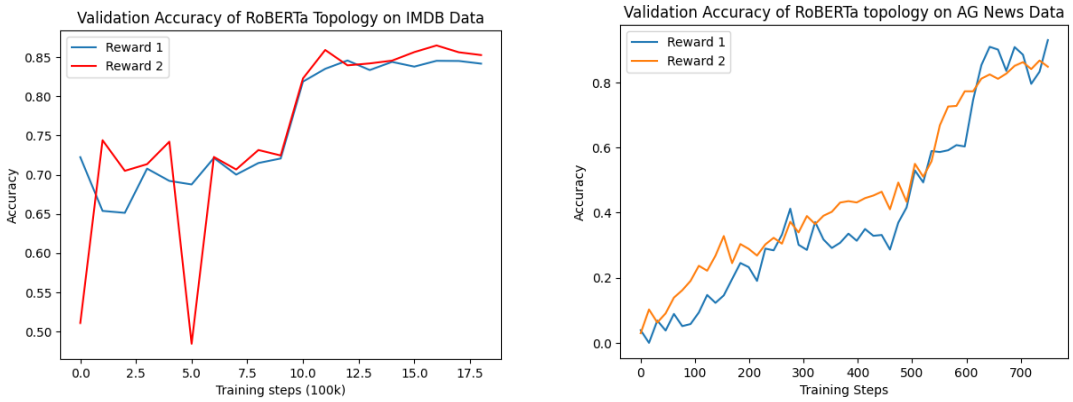


Figure 5.11 RoBERTa Topology's training performance on AG News dataset (left to right: Accuracy, Precision, Recall, F1 Score)



(a) Performance on IMDB

(b) Performance on AG News

Figure 5.12 RoBERTa + LSTM topology performance on two datasets' validation split

RoBERTa + LSTM topology attains 0.84 ± 0.008 accuracy on the test set of IMDB and 0.9 ± 0.004 accuracy on the test set of AG News.

5.3.3 Overview of the Model Performances

Here we give the model performances on the test datasets, when trained with two different reward functions. We demonstrate these on Table 5.1.

In our examination of CNN + LSTM topology, policies predominantly converge to *Full-Read Tendency* with reward function R_2 and *First-Chunk Focus* with R_1. However, in rare instances, *Skip-Twice Strategy* is encountered with R_2, yielding optimal results on the IMDB. With DistilBERT + LSTM, the prevalent tendency of policies is towards *First-Chunk Focus*, potentially attributable to the inherent potency of DistilBERT. While reward functions R_1 and R_2 generate similar behaviors on AG News, distinctions arise on IMDB; where, occasional emergence of *Skip-Twice Strategy* with R_1 and *Full-Read Tendency* with R_2. This divergence might be influenced by the penalties associated to the reading more text. For RoBERTa + LSTM, policies consistently converge to either *First-Chunk Focus* or *Full-Read Tendency*, regardless from the reward function. Notably, *First-Chunk Focus* is predominant on AG News, due to its simplicity, while *Full-Read Tendency* is more common on IMDB, where classification task is more challenging as the users tend to write long irrelevant sentences in their reviews.

Topology	**Reward Function**	**Best Accuracy on IMDB (whole reading model accuracy) / Policy Behaviour**	**Best Accuracy on AG News (whole reading model accuracy) / Policy Behaviour**
CNN + LSTM	R_1	0.76 (0.82) / *Skip-Twice Strategy*	0.86 (0.91) / *First-Chunk Focus*
DistilBERT + LSTM	R_1	0.81 (0.89) / *Skip-Twice Strategy*	**0.92** (0.95) / *First-Chunk Focus*
RoBERTa + LSTM	R_1	**0.84** (0.9) / *Full-Read Tendency*	0.9 (0.94) / *First-Chunk Focus*
CNN + LSTM	R_2	0.75 (0.82) / *Full-Read Tendency*	0.86 (0.91) / *Full-Read Tendency*
DistilBERT + LSTM	R_2	0.82 (0.89) / *Full-Read Tendency*	**0.91** (0.95) / *First-Chunk Focus*
RoBERTa + LSTM	R_2	**0.85** (0.9) / *Full-Read Tendency*	0.9 (0.94) / *First-Chunk Focus*

Table 5.1 Performances of the topologies on test sets

Topology	**Policy Behaviour**	**# FLOPS per one forward pass**	**Training time per epoch (20k samples)**	**Inference time (5k samples)**
CNN + LSTM	*First-Chunk Focus*	1.59 million	890s	85s
CNN + LSTM	*Full-Read Tendency*	31.98 million	1450s	131s
CNN + LSTM	*Skip-Twice Strategy*	7.99 million	1020s	99s
CNN + LSTM	whole reading model	39.58 million	2133s	304s
DistilBERT + LSTM	*First-Chunk Focus*	0.85 billion	1948s	243s
DistilBERT + LSTM	*Full-Read Tendency*	17.18 billion	3684s	450s
DistilBERT + LSTM	*Skip-Twice Strategy*	4.29 billion	2775s	311s
DistilBERT + LSTM	whole reading model	17.18 billion	4770s	514s
RoBERTa + LSTM	*First-Chunk Focus*	1.71 billion	4226s	414s
RoBERTa + LSTM	*Full-Read Tendency*	34.18 billion	8753s	676s
RoBERTa + LSTM	*Skip-Twice Strategy*	8.55 billion	6644s	591s
RoBERTa + LSTM	whole reading model	34.20 billion	11385s	999s

Table 5.2 Training/Inference Time and FLOP statistics of the topologies

We also inspect the required FLOPs and training and inference time of our topologies in Table 5.2. When we investigate the number of FLOPs per one forward pass, that is, the FLOPs required for the topology to complete one episode, we see that we have achieved to decrease this cost by around 95% with the policies converging to *First-Chunk Focus* and by around 80% with the policies converging to *Skip-Twice Strategy*, when compared to the whole reading models constructed with the same topology. This is not an unexpected outcome as policies with these behaviours observe shorter samples of text compared to the whole reading model. We even observe around 20% decrease in FLOPs for CNN + LSTM topology and 1% decrease in RoBERTa + LSTM topology with *Full-Read Tendency*, which seems to observe texts of the same length as the whole reading model.

We posit that, enhancements for both training and inference times in our topologies that converge to any behaviour is due to the reduction in the number of parameters used in the calculations for forward and backward passes. These results show that together with the decreased number of FLOPs, the topologies were able to introduce faster training and inference.

5.4 Conclusion and Future Work

In conclusion, our study aimed to reproduce the results reported in Yu et al. (2018) but encountered challenges in achieving identical outcomes. However, our findings align closely with other institutions (Xie et al., 2019) that attempted to replicate the results, suggesting the presence of common factors affecting the reproducibility of the original study. Notably, by leveraging transformers and pretrained Language Models (LLMs) as feature extractors and making use of another reward function, we improved the accuracy of our results compared to the initial findings.

- We showcased the effectiveness of our policies, that obtain the behaviours *First-Chunk Focus* and *Skip-Twice Strategy* that does not require to observe the entirety of text to make a classification.
- We accomplished to reduce the FLOPs on average 60% in training and inference for all the policy behaviours we obtained. Meanwhile, we achieved to preserve the accuracy by only suffering on average 5% loss.
- We show that train and inference times are reduced in all behaviours, with

60% for *First-Chunk Focus*, 46% for *Skip-Twice Strategy* and 25% for *Full-Read Tendency* on average.

To further advance the field, we propose future work that explores the use of transformers with gating mechanisms as an episodic memory component demonstrated in Parisotto et al. (2019), replacing LSTM-based approaches. We believe this shift in architecture could potentially yield significant improvements in performance and learning capabilities.

Additionally, training the policies using Proximal Policy Optimization (PPO) coupled with Reinforcement Learning from Human Feedback as in (Ramamurthy et al., 2023) can contribute to enhanced stability during the training process.

It is worth noting that the inability to replicate the exact results might stem from undisclosed implementation details in the original study, such as the dimension of the word embeddings, usage of pretrained word embeddings, dimensions of the LSTM, or the number of training steps employed.

BIBLIOGRAPHY

Bellman, R. E. (2003). *Dynamic Programming*. Dover Publications, Inc., USA.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. In *Journal of machine learning research*.

Bradley Knox, W. and Stone, P. (2008). Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE International Conference on Development and Learning*, pages 292–297.

Chai, D., Wu, W., Han, Q., Wu, F., and Li, J. (2020). Description based text classification with reinforcement learning.

Chan, H. P., Chen, W., Wang, L., and King, I. (2019). Neural keyphrase generation via reinforcement learning with adaptive rewards.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2023). Deep reinforcement learning from human preferences.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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

Dulac-Arnold, G., Denoyer, L., and Gallinari, P. (2011). Text classification: A sequential reading approach. In *Lecture Notes in Computer Science*, pages 411–423. Springer Berlin Heidelberg.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202.

Goldberg, Y. and Hirst, G. (2017). *Neural Network Methods in Natural Language Processing*. Morgan amp; Claypool Publishers.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Johnson, R. and Zhang, T. (2015). Effective use of word order for text categorization with convolutional neural networks.

Kim, Y. (2014). Convolutional neural networks for sentence classification.

Kreutzer, J., Riezler, S., and Lawrence, C. (2020). Offline reinforcement learning from human feedback in real-world sequence-to-sequence tasks.

Lagoudakis, M. G. and Parr, R. (2003). Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, H. and Xu, H. (2020). Deep reinforcement learning for robust emotional classification in facial expression recognition. *Knowledge-Based Systems*, 204:106172.

Lin, E., Chen, Q., and Qi, X. (2020). Deep reinforcement learning for imbalanced classification. *Applied Intelligence*, 50(8):2488–2502.

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.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

MacGlashan, J., Ho, M. K., Loftin, R. T., Peng, B., Roberts, D. L., Taylor, M. E., and Littman, M. L. (2017). Interactive learning from policy-dependent human feedback. *CoRR*, abs/1701.06049.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mansour, R. F., Escorcia-Gutierrez, J., Gamarra, M., Villanueva, J. A., and Leal, N. (2021). Intelligent video anomaly detection and classification using faster rcnn with deep reinforcement learning model. *Image and Vision Computing*, 112:104229.

Mao, Y., Qu, Y., Xie, Y., Ren, X., and Han, J. (2020). Multi-document summarization with maximal marginal relevance-guided reinforcement learning.

Mao, Y., Tian, J., Han, J., and Ren, X. (2019). Hierarchical text classification with reinforced label assignment. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Martinez, C., Ramasso, E., Perrin, G., and Rombaut, M. (2020). Adaptive early classification of temporal sequences using deep reinforcement learning. *Knowledge-Based Systems*, 190:105290.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

OpenAI (2023). Gpt-4 technical report.

Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan. Association for Computational Linguistics.

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N., and Hadsell, R. (2019). Stabilizing transformers for reinforcement learning.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Qian, X., Zhong, Z., and Zhou, J. (2018). Multimodal machine translation with reinforcement learning.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

Ramamurthy, R., Ammanabrolu, P., Brantley, K., Hessel, J., Sifa, R., Bauckhage, C., Hajishirzi, H., and Choi, Y. (2023). Is reinforcement learning (not) for natural language processing: Benchmarks, baselines, and building blocks for natural language policy optimization.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Satija, H. and Pineau, J. (2016). Simultaneous machine translation using deep reinforcement learning.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Wang, J., Sun, C., Li, S., Wang, J., Si, L., Zhang, M., Liu, X., and Zhou, G. (2019). Human-like decision making: Document-level aspect sentiment classification via hierarchical reinforcement learning.

Watkins, C. (1989). Learning from delayed rewards.

Wiering, M. A., van Hasselt, H., Pietersma, A.-D., and Schomaker, L. (2011). Reinforcement learning algorithms for solving classification problems. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 91–96.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256.

Wu, L., Tian, F., Qin, T., Lai, J., and Liu, T.-Y. (2018). A study of reinforcement learning for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3612–3621, Brussels, Belgium. Association for Computational Linguistics.

Xia, Y., He, D., Qin, T., Wang, L., Yu, N., Liu, T.-Y., and Ma, W.-Y. (2016). Dual learning for machine translation.

Xie, F., Deng, W., and Liu, P. (2019). Reproduction of "fast and accurate text classification" paper from iclr 2018. `https://github.com/COMP6248-Reproducability-Challenge/Fast-And-Accurate-Text-Classification-Reproduction/tree/master`.

Xu, J., Zhao, L., Yan, H., Zeng, Q., Liang, Y., and Sun, X. (2019). LexicalAT: Lexical-based adversarial reinforcement training for robust sentiment classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5518–5527, Hong Kong, China. Association for Computational Linguistics.

Yang, P., Ma, S., Zhang, Y., Lin, J., Su, Q., and Sun, X. (2018a). A deep reinforced sequence-to-set model for multi-label text classification. *CoRR*, abs/1809.03118.

Yang, P., Sun, X., Li, W., Ma, S., Wu, W., and Wang, H. (2018b). Sgm: Sequence generation model for multi-label classification.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Yu, A. W., Lee, H., and Le, Q. V. (2017). Learning to skim text.

Yu, K., Liu, Y., Schwing, A. G., and Peng, J. (2018). Fast and accurate text classification: Skimming, rereading and early stopping.

Zhang, X., Zhao, J. J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *NIPS*.

Zhao, D., Chen, Y., and Lv, L. (2017). Deep reinforcement learning with visual attention for vehicle classification. *IEEE Transactions on Cognitive and Developmental Systems*, 9(4):356–367.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.

Zhu, Z., Wang, L., Magnier, B., Zhu, L., Zhang, D., and Yu, L. (2022). Reinforcement learning driven intra-modal and inter-modal representation learning for 3d medical image classification. In Wang, L., Dou, Q., Fletcher, P. T., Speidel, S., and Li, S., editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*, pages 604–613, Cham. Springer Nature Switzerland.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P. F., and Irving, G. (2019). Fine-tuning language models from human preferences. *CoRR*, abs/1909.08593.