THE UNIVERSITY OF AUCKLAND

MASTER OF SCIENCE THESIS

Exploring Neuromodulation in General Question Answering

Author:
Kobe KNOWLES

Supervisor:
Dr. Joshua BENSEMANN
Dr. Lia LEE

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Strong AI Lab
School of Computer Science

July 17, 2022

THE UNIVERSITY OF AUCKLAND

Abstract

Faculty of Science
School of Computer Science

Master of Science

Exploring Neuromodulation in General Question Answering

by Kobe KNOWLES

There exist problems with a QA model's ability to generalise to unseen data. Current QA models perform well when trained on a dataset individually, but they often perform worse on other datasets of the same task [105]. In the current QA paradigm, QA models are almost exclusively Transformers; therefore, we aim to improve the Transformer's generalisation capabilities. In an attempt at such, we introduce the Neuromodulated Transformer (NeMoT): an extension to the Transformer via the entwinement of neuromodulation. We hypothesise that the addition of neuromodulation, when coupled with an environment that encourages it (e.g., multi-task and multi-format learning), will result in better generalisation capabilities. We show that NeMoT showcases better generalisation capabilities than a baseline model of a similar structure and 65 million more parameters; however, further experiments are needed to reach a definitive conclusion.

In QA, the ability to comprehend text is essential to the generation of the correct answer. Reading strategies are utilised by humans to help them comprehend text and improve their reading proficiency; they have been integrated into QA models in an attempt to achieve the same effects. As a secondary objective in this thesis, we modify and integrate the answer option interaction reading strategy [109] and highlighting reading strategy [86] with NeMoT. Results show that incorporating reading strategies with NeMoT improves performance on MQA datasets.

Acknowledgements

I am extremely grateful to my supervisors Dr. Lia Lee and Dr. Joshua Bensemann for their continual support, guidance, and excellent feedback over the last year during my Masters degree. I have improved immensely under their wings as a researcher.

I am also extremely grateful to my family who have given me immense support throughout my university years, especially during my Masters degree.

Contents

List of Figures

List of Tables

List of Abbreviations

SOTA	**S**tate **O**f **T**he **A**rt
NLP	**N**atural **L**anguage **P**rocessing
QA	**Q**uestion **A**nswering
GQA	**G**eneral **Q**uestion **A**nswering
MQA	**M**ultiple-choice **Q**uestion **A**nswering
NeMoT	**Ne**uro**Mo**dulated **T**ransformer
AOI	**A**nswer **O**ption **I**nteraction
POS	**P**art-**o**f-**s**peech
ANN	**A**rtificial **N**eural **N**etwork

Chapter 1

Introduction

Natural language processing (NLP) is an important field in machine learning, evermore so with the abundance of text available in digital format and the speed at which machines can process large amounts of text. NLP entails the processing and analysis of text with the goal of extracting information to perform specific tasks. Question answering (QA) is one such task, where given a question a model is tasked with generating an answer or picking one of the provided answers to the question. To answer a question correctly a model needs the ability to comprehend, reason, and utilise internal knowledge about the world (i.e., perform inference). A question can be formulated in many ways such as:

- **Question**: a question alone is given as input to the model. The model is expected to generate an answer to the question with no other context, utilising only the internal parameters of the model.
- **Question + answer options**: a question is given with a set of answer options to the model; only one answer option is usually correct. The model is expected to choose what it deems the correct answer to the question from the set of provided answers, utilising only the internal parameters of the model.
- **Passage + question**: a passage and question are given as input to the model. The model is expected to generate an answer to the question utilising the passage and internal parameters of the model.
- **Passage + question + answer options**: a passage, question, and a set of answer options are given as input to the model; only one answer option is usually correct. The model is expected to choose what it deems the correct answer to the question from the set of provided answers, utilising the passage and internal parameters of the model.

In this thesis, we fixate on the QA subset of NLP, especially since NLP tasks can be posed as questions [56]. The ability of current QA models to generalise to other QA datasets that it has not been trained on is lacking, especially since, given a task such as reading comprehension, for example, there is a gap in performance between a training dataset of that task and other datasets — that the model has not seen during training — of the same task [105]. We denote the task where the model is expected to generalise to out-of-domain datasets with no further training as *general question answering* (GQA), which is the primary focus of this thesis.

We focus on changes to the Transformer architecture [90] specifically, in an attempt to improve its generalisation capabilities in QA. Other techniques to improve performance in GQA include: multi-task training procedures [49, 87, 105], the text-to-text framework [41, 42, 68], etc. Two ideas we consider are neuromodulation

and reading strategies. Neuromodulation refers to a process that involves the regulation of a population of neurons in a context dependent manner [40]. Reading strategies refer to the planned and explicit actions that a reader undertakes to help them decipher text to meaning [61, 71]; they are utilised to improve their reading comprehension and proficiency[1].

1.1 Motivation

Given any potential question that could be posed to a QA model, the ultimate goal is for the ability to answer any question correctly if it is possible. The consequences of such a system are that it would currently top the leaderboards of all QA datasets and would surpass the performance of humans. This entails a model that can generalise to out-of-domain (i.e., unseen) questions that it has not encountered before. Therefore, achieving good performance in GQA is a stepping stone towards achieving such a system. The real-world applications of such an accomplishment include aiding researchers by automating redundant tasks, an improved question answering ability of search engines and dialogue systems, medical diagnosis, and possibly a step towards AGI among others, all of which have the potential to provide a positive impact to the world.

The current state of GQA in the literature includes: benchmarks to evaluate performance [41, 42, 93, 94], training procedures that generally involve multi-task training [49, 87], and a text-to-text framework that is utilised to unify differing QA formats [41, 42, 68]. The Transformer architecture is the current SOTA in terms of performance in QA [37, 41, 42, 52, 100, 110, 111]. The foundational architecture of the Transformer is the vanilla Transformer [90], where its descendants modify it in ways that may or may not be better for generalisation. In this thesis, we make modifications to the Transformer architecture intending to improve performance in QA and GQA. Specifically, for neuromodulation we focus on GQA, while for reading strategies we focus only on performance in QA (i.e., performance on individual datasets).

Reading strategies refer to the planned and explicit actions undertaken by an individual to help them decipher text to meaning, or in other words, improve their reading comprehension and proficiency in reading [61, 71]. Neuromodulation refers to a biological mechanism that is involved in the continuous tuning of a neuron's input and output behaviour conditioned on external stimuli in a context-dependent manner [5, 54, 91]. We hypothesise that the insertion of reading strategies will allow for better reading comprehension and proficiency, allowing the model to perform better in QA. Additionally, we hypothesise that neuromodulation will allow for the modification of the forward propagation through the network in a context-dependent manner, allowing for more complex rules to be learned, that when coupled with an environment that encourages generalisation, will result in better performance in GQA. Therefore, in this thesis, we focus on evaluating neuromodulation's potential in GQA as a primary objective and the performance of reading strategies in QA as a secondary objective.

[1] Our code can be found at https://github.com/Strong-AI-Lab/Neuromodulated-Transformer

1.2 Problem description

We are interested in the impact that the entwinement of neuromodulation with the Transformer architecture has in GQA, and additionally, the impact that the incorporation of reading strategies with the Transformer architecture has in QA. We explore the following research questions:

- Does the entwinement of neuromodulation with the Transformer architecture improve its QA capabilities on individual datasets?
- Does the entwinement of neuromodulation with the Transformer architecture, when coupled with an environment that encourages it, allow for better generalisation in QA?
- Can the utilisation of reading strategies improve performance in MQA?

1.3 Objectives

Our research focuses on improving the generalisability and performance of QA models. We are interested in integrating reading strategies and neuromodulation with the Transformer architecture. The aims of our research are:

- To entwine neuromodulation with the Transformer architecture to improve its generalisation capabilities.
- To incorporate reading strategies with the Transformer architecture to improve performance in MQA.

1.4 Contributions

Our main contributions are as follows:

- An extension to the Transformer, the Neuromodulated Transformer (NeMoT): a simple entwinement of the Transformer architecture and neuromodulation. To our knowledge, this is the first model to explicitly incorporate neuromodulation with the Transformer architecture and to test its generalisation capabilities. Previous work in neuromodulation is primarily in the continual learning domain and is applied to the individual connections between neurons or is incorporated into backpropagation as we have done. We believe that there is potential for neuromodulation in the non-continual learning domain (i.e., where the model is fixed after training) and that the incorporation of it will potentially allow for better performance and generalisation.
- An extension to two existing reading strategies: answer option interaction (AOI) [109] and highlighting [86]. They are both incorporated into NeMoT and their performance on MQA datasets is measured. Previous work on reading strategies involves incorporating them into ANNs, which generally results in an improved performance; however, the AOI and highlighting reading strategies, in particular, are not perfect. We extend the two reading strategies in an attempt to overcome their shortcomings. AOI computes a bilinear representation between each combination of answer options, which in comparison to a version where it does not have to sequentially go through each combination,

is computationally inefficient. We extend it by simplifying it, only computing a bilinear representation once for all answer options together. Highlighting is quite simple and adds one of two vectors to each token in a document embedding: ℓ^+ if the token's POS tag is a noun, verb, adjective, adverb, numerical, or foreign word; ℓ^- otherwise. We extend it by having a unique vector for each of the following POS tags: noun, verb, adjective, adverb, numerical, and foreign word.

1.5 Overview of research

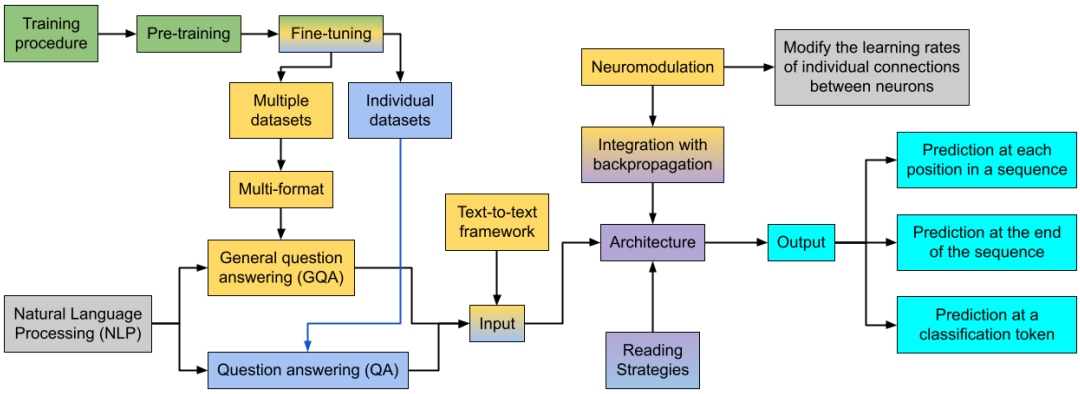


FIGURE 1.1: An overview of the entire process of constructing a QA system.

Figure 1.1 provides an overview of the procedure for constructing a QA model; it is the procedure we use to construct QA models in this thesis. Our research focuses on improving the architecture, more specifically, the Transformer architecture [90]. We experiment with entwining neuromodulation and incorporating reading strategies with it. Precisely, we are interested in the impact neuromodulation has on generalisation capabilities. We lower the scope for reading strategies and are only interested in analysing its contribution to performance on MQA datasets, training on each individually — however, we might expect performance to be better in the GQA domain as well, even though we do not test for it. In this thesis, all other components in the procedure are taken from the literature and are largely left unchanged.

1.6 Structure of thesis

This thesis consists of the following chapters:

- Chapter 2: Background.
  In this chapter we introduce reading strategies, metacognition, and neuromodulation. We define what they are and detail relevant literature in regards to this thesis's topic, QA. We dive deep into the Transformer architecture [90] and give an overview of some descendants of the architecture (i.e., BERT and GPT). We define what general question answering is and portray the current literature. Lastly, we introduce a wide variety of QA datasets and their metrics to evaluate performance; we utilise them in this thesis.

- Chapter 3: A Simple Entwinement of Neuromodulation and the Transformer.
  In this chapter we introduce the Neuromodulated Transformer (NeMoT), an extension to the Transformer via the entwinement of neuromodulation. Additionally, we describe extensions to two reading strategies and incorporate them into NeMoT. We detail the training procedure and conduct a variety of experiments in QA to measure the performance of NeMoT versus a baseline, including if it resulted in an improved generalisation (i.e., better performance in GQA). Furthermore, we conduct experiments with two reading strategies on MQA datasets to see if their presence resulted in an improved performance.

- Chapter 4: Conclusion.
  We conclude this thesis by stating the achievements of our work, answering the research questions, discussing the limitations, and we provide potential avenues for future work.

Chapter 2

Background

This chapter covers all relevant literature needed for an intuitive understanding of this thesis. Sections 2.1, 2.2, and 2.3 cover reading strategies, metacognition, and neuromodulation, respectively. Section 2.4 details the Transformer architecture. Section 2.5 highlights the current state of general question answering. Lastly, Section 2.6 displays the datasets utilised in this thesis.

2.1 Reading strategies

2.1.1 Defining reading strategies

With the abundance of information stored physically in books, digitally in ebooks and on the web, the ability for humans to efficiently extract knowledge from the abundance of text is crucial in today's day in age. The process of achieving this is known as reading comprehension: a process where meaning is constructed through coordinating processes such as reading, knowledge of words, reading strategies, and knowledge of the world among others [71]. The reader is to utilise their internal knowledge of the world and combine it with what is read to construct meaning and update their internal knowledge of the world.

Metacognitive reading strategies are one such idea that can be taught to help improve an individual's reading comprehension capabilities, i.e., their ability to understand and decipher text to meaning [61, 71]. Several studies indicate that there is a strong positive relationship between the use of metacognitive reading strategies by an individual and their reading comprehension capabilities [14, 71, 76, 79, 108].

There exist differences between skilled and novice readers metacognitively: "skilled readers often engage in deliberate activities that require planful thinking, flexible strategies, and periodic self-monitoring" [63], while novice readers are blind to the metacognitive component and all that that entails [61, 63]. One observed example of this is that a good reader focuses on the text as a whole, while a novice reader tends to focus on the meaning of individual words [4, 61, 71]. The metacognitive component of metacognitive reading strategies has been split into a separate section (see Section 2.2) while the rest of this section focuses on defining reading strategies solely.

Reading strategies is an expansive term used to characterise the planned and explicit actions that a reader undertakes to help them decipher text to meaning [61, 71]. They are utilised by an individual to improve their reading comprehension and proficiency in reading. They are often taught to individuals who lack reading comprehension capabilities (e.g. second language learners and children). "Previewing text before reading", "guessing the meaning of unknown words" and "summarizing text information" are some examples of reading strategies [106].

A reading strategy can be split into three different types: *global* reading strategies, *problem solving* reading strategies, and *support* reading strategies [61, 106]. A global reading strategy represents intentional, carefully planned techniques that are directed toward a global analysis of the text, allowing a reader to manage and monitor their reading. Examples include "checking how the text content fits purpose" and "previewing the text before reading". A problem solving reading strategy represents localised, focused techniques that are used to remedy issues that arise in the understanding of text. Examples include "adjusting the reading rate", "re-reading for better understanding", and "guessing the meaning of unknown words". A support reading strategy represents mechanisms that aid the reader's understanding of the text. Examples include "using reference materials" and "discussing reading with others". See Table 2.1 for a list of all reading strategies and their type.

Type	Strategy	Type	Strategy
Global	Setting the purpose for reading	Problem	Reading slowly and carefully
Global	Using prior knowledge	Problem	Trying to stay focused while reading
Global	Previewing text before reading	Problem	Adjusting reading rate
Global	Checking how text content fits purpose	Problem	Paying close attention to reading
Global	Skimming to note text characteristics	Problem	Pausing and thinking about reading
Global	Determining what to read	Problem	Visualizing information read
Global	Using text features (e.g., tables)	Problem	Re-reading for better understanding
Global	Using context clues	Problem	Guessing the meaning of unknown words
Global	Using typographical aids (e.g., italics)	Support	Taking notes while reading
Global	Critically evaluating what is read	Support	Reading aloud when text becomes hard
Global	Resolving conflicting information	Support	Summarizing text information
Global	Predicting or guessing text meaning	Support	Discussing reading with others
Global	Confirming predictions	Support	Underlining information in text
Support	Using reference materials	Support	Paraphrasing for better understanding
Support	Going back and forth in text	Support	Asking oneself questions

TABLE 2.1: All 30 reading strategies and whether or not they are a global (Global), problem solving (Problem) or support (Support) reading strategy [106].

2.1.2 Reading strategies used in machines for QA

Given the utilisation of reading strategies to improve a reader's comprehension and proficiency, the same may carry over to QA models. They are seldom used by QA models but have shown to improve performance when they are utilised. We highlight the utilisation of reading strategies in QA models in this section.

Dual Co-Matching Network

The Dual Co-Matching Network (DCMN) utilises two reading strategies to help improve the performance of their model in MQA [109]. The two reading strategies are answer option interaction (AOI) and passage sentence selection (PSS).

AOI allows for each answer option to be compared directly with one another, where like in humans, it may be used to eliminate incorrect answer options; if the model is required to make a random guess as it does not know the correct answer, eliminating some answer options from the fray will result in a better performance on average because it has a higher chance of picking the correct answer option. They build bilinear representations between each combination of answer options and use a gated mechanism to merge them back together.

Passage sentence selection allows for the selection of the most salient passages conditioned on the question and a specific answer option. Each passage is scored via the cosine score or bilinear score; the top k passages with the highest score are selected.

General reading strategies

Three general reading strategies are utilised in [86], with all producing an improvement in performance: back and forth reading (BF), highlighting (HL), and self-assessment (SA). BF simply involves training two identical models: the first's input consists of the answer option, question, and document in that order; the second's input consists of the document, question, and answer option in that order (i.e., it is reversed). Both models are combined via an ensemble method. HL involves adding to a document embedding one of two vectors for each token in the document embedding: ℓ^+ if the POS at that position is a noun, verb, adjective, adverb, numeral, or foreign word; ℓ^- otherwise. SA refers to training the model on a set of automatically generated questions and answer options from a reference document.

Incorporating reading strategies into a document-based QA system

A document-based question answering system is created in [46] that relates to the sequential process of three reading strategies. Given a document and question the three reading strategies are: the skimming of the document to acquire general knowledge about it; the reading of the question carefully, utilising the generally acquired knowledge via the skimming of the document; the traversal of the document again to generate an answer to the previously read question. The architecture of the system explicitly encodes the previously described procedure into the architecture in hopes of having the same effects that it does for humans.

In the first step either the title of the document or an extracted summary from a summary extraction method is utilised. Both the question and summary are encoded via an RNN, where directly after the two representations are merged, completing the second step. A hierarchical RNN is then utilised to capture dependencies between sentences and to generate an answer, completing the third step.

Incorporating reading strategies into abstractive summarization

A three-phased attack is utilised in [101] to incorporate human-like reading strategies into abstractive summarization. The first phase involves the acquisition of a general understanding of a document via a knowledge-attention network. The second phase involves multi-task learning, where the tasks learned help the model acquire skills relevant for the generation of a summary. The third phase is a polishing process via a generative adversarial network (GAN).

2.2 Metacognition

To define metacognition, it is first essential to define cognition. Cognition refers to the mental actions and processes involved in knowing and awareness, such as the ability to comprehend through thought, experiences, and stimuli among others [64]. The processes involved in cognition are referred to as cognitive processes: they include comprehension, attention, memory, reasoning and problem solving among others [96].

John Flavell introduced metacognition in the early 1970s, building upon the term metamemory of previous work [4]. Metacognition is defined as knowledge about cognition and regulation (i.e., control) of cognition, or alternatively, simply thinking about thinking. It is considered a high-level cognition that can be split into many sub-components, but of two mainly: knowledge and control [4].

Metacognitive knowledge refers to an individual's knowledge about how their cognition operates, including what strategies to use for a given domain [35]. This includes knowledge of person variables (i.e., what humans are like as cognitive beings), task variables (i.e., how certain information encountered influences and constricts how an individual deals with it), and strategy variables (i.e., cognitive strategies for achieving goals). In the context of strategy use, it includes the knowledge of what the strategies for a given domain are and when, where, and why different strategies should be utilised.

Metacognitive control refers to how an individual controls their cognitive process, which includes the monitoring of said processes [4]. It entails self-regulation and executive functioning. Self-regulation refers to "the ability to control one's own behaviours and cognitive activities" [4] and executive functioning refers to "the cognitive system that controls and manages other cognitive processes" [4]. In the context of metacognitive reading strategies (see Section 2.1), this entails the knowledge of reading strategies and the continual monitoring and control of the strategies and cognitive processes involved.

There exist various approaches to assess someone's metacognitive ability: verbal reports, online processing measures, and judgement of learning or predictions of outcome [4].

Verbal reports are the primary way to collect information about metacognitive knowledge. Individuals are asked what they know (i.e., their knowledge) and what they do (i.e., control) when engaging in cognitive tasks. A major concern with this approach is that what people say may not correspond with what they do.

Online processing measures are for the purpose of measuring the control aspect of metacognition. An individual is asked to engage in a task, while processing collecting measures (e.g., computer technology that tracks eye movements) are carried out to collect relevant information. This rectifies the issue with verbal reports as we know accurately what the person is doing; however, the technological component may disrupt the naturalness of the task.

Lastly, judgement of learning (JOL) and predictions of performance (POP) refer to an individual's accuracy in predicting their learning capabilities and their performance on a provided task, respectively. Additionally, feeling of warmth (FOW) [35] is a similar technique where during the problem-solving process individuals are asked to provide how close they are to a solution (e.g., varying levels of hot and cold if they are close and far away from a solution, respectively). In all cases, the more accurate they are, the more skilled metacognitively they are said to be.

2.3 Neuromodulation

2.3.1 Neuromodulation in organisms

Neurons are a fundamental building block of the brain and central nervous system. The main components of a neuron are the potentially thousands of dendrites, the cell body, and the axon (we utilise [8, 85] as our sources in this paragraph). The dendrites are involved in receiving signals from other neurons; the cell body manufactures and recycles proteins; and the axon is involved in the facilitation of the outgoing

signal to other neurons through a synapse. A synapse is a connection between two neurons and the gap between the two neurons is referred to as the synaptic cleft. The communication of neurons through a synapse is done through either chemical or electrical signals; the process is referred to as neurotransmission. The chemical signal released into the synaptic cleft is known as a neurotransmitter, it binds to receptors on target cells (neurons), changing their electrical properties and resulting in a large variety of post-synaptic effects [17].

Neural plasticity refers to "the ability of neurons to change form and function in response to alterations in their environment" [38, 81] and is an important aspect of lifelong learning, through its strong contribution to adaptation and learning in biological neural networks. Synaptic plasticity refers to changes that occur at synapses, modifying how neurons communicate with one another [9]. One form of synaptic plasticity is Hebbian plasticity: it is activated by and further magnifies correlations in neural activity [107]. An important part of neural plasticity is neuromodulation [81]: a biological mechanism that is involved in the continuous tuning of a neuron's input and output behaviour conditioned on external stimuli in a context dependant manner [5, 54, 91]; it entails the ability to change the learning rate of individual connections between neurons [26]; and it plays a key role in the facilitation of learning [22].

Specifically, neuromodulation refers to a subset of neurotransmitters called neuromodulators [5, 54, 91]. They are chemical signals with the ability to locally modify learning rates by either up-regulating or down-regulating them in response to external stimuli [1, 11, 26, 36]. They have spatially distributed and temporally extended effects on recipient neurons, potentially allowing for the regulation of a population of neurons [24, 39, 40, 55, 75]. Examples of neuromodulators are dopamine, serotonin, noradrenaline, and acetylcholine [24].

2.3.2 Neuromodulation in machine learning

Neuromodulation has been incorporated into ANNs in a variety of ways, mostly in the continual learning domain. They are utilised to modify the learning rates of weights in individual circumstances, by varying the strength of the learning signal; in the biological sense this is the modulation of Hebbian learning, the strengthening or weakening of connections between neurons [19, 20, 26, 82, 92]. They are often coupled with an evolutionary algorithm to search for the initial parameters of a network and their topologies, of which the weights are then updated according to update rules involving neuromodulation, not backpropagation; in some instances, the error generated by the model is incorporated into the weight updates, but it generally is not. More recent work on neuromodulation involves integrating it with the current dominant learning algorithm, backpropagation [6, 58, 83, 91]. In this section, we only detail the neuromodulation aspects of each paper.

Incorporating neuromodulation without backpropagation

Modulatory Neurons are introduced into ANNs by [82], where they fulfill the purpose of either boosting or curtailing neural plasticity at target neurons, allowing learning to occur at specific parts of a network conditioned on various different scenarios (i.e., context). Specifically, the contribution of the paper is the introduction of *modulatory neurons* that act alongside *standard neurons*. For all neurons in a network, each contains a *standard activation* a_i and *modulatory activation* m_i for the ith neuron,

for example. The output of each neuron is denoted by o_i for the ith neuron and is calculated by $o_i = \tanh(a_i/2)$.

For the ith neuron the standard activation is calculated by

$$a_i = \Sigma_{j \in Std}\, w_{j,i} \cdot o_j,$$

and that for the modulatory activation by

$$m_i = \Sigma_{j \in Mod}\, w_{j,i} \cdot o_j,$$

where $w_{j,i}$ is the connection strength between the jth and ith neuron, and Std is the set of standard neurons and Mod is the set of modulatory neurons.

The modulatory activation has no contribution to the output of the neuron, but only to the update of $w_{j,i}$. It is updated according to

$$\Delta w_{j,i} = \tanh(m_i/2) \cdot \delta_{j,i},$$

where $\delta_{j,i}$ is the plasticity term and $\tanh(m_i/2)$ references the modulation value. The modulation value is a real number in the range of -1 and 1 that modulates the plasticity term: if it is zero then no learning occurs, if it is negative then it inverts the plasticity term, and the further the value is away from zero, the higher the rate of learning.

The plasticity term is defined as

$$\delta_{j,i} = \eta \cdot (Ao_jo_i + Bo_j + Co_i + D),$$

where η is the learning rate, o_j is the output of the jth neuron (which is input to the ith neuron), o_i is the output of ith neuron, and the following four tunable parameters: the correlation term A, the input (pre-synaptic) term B, the output (post-synaptic) term C, and the constant term D.

The plasticity term is simplified in [26], where $\delta_{j,i}$ only consists of a Hebbian learning term $o_j \cdot o_i$ and the learning rate η. When o_j and o_i are correlated the Hebbian learning term is large; the less correlated they are, the smaller the term is.

Furthermore, [92] modifies the original architecture further by introducing *point sources*, which replaces the modulatory neurons; they call the resulting mechanism diffusion-based neuromodulation. Point sources are added to ANNs at specific locations with their role being to introduce diffusing learning signals to the network associated with the task being learned.

Diffusion-based neuromodulation involves the weight update rule

$$\Delta w_{j,i} = \eta \cdot m_i \cdot o_j \cdot o_i,$$

where η, m_i, o_j, and o_i are the learning rate, modulatory factor, output for neuron j, and output for neuron i, respectively. The modulatory factor of neuron i for a specific problem formulation provided in their paper is

$$m_i = \sigma(o_s g(d_{is}) + o_w g(d_{iw})),$$

where o_s and o_w refer to the output of the two point sources (each representing summer and winter, respectively, in the paper's toy problem), g is a Gaussian function, σ is a sigmoid function, and d_{is} and d_{iw} is the distance of a neuron from the summer and winter points, respectively. If neuron i is within a distance of 1.5 from a

point source, then the contribution to the modulatory factor increases according to g; otherwise, the contribution is 0. For a full description, we refer readers to [92].

ModNet is an architecture that is inspired by the mushroom body of insects [19, 20] and is quite similar in some aspects to [82], which introduces modulatory neurons. ModNet consists of an input layer, a hidden layer, an output layer, and a modulatory layer. The output layer consists of standard activations and modulatory activations.

The standard activations are defined as

$$a_i = \sigma(\Sigma_{j \in Hid} w_{j,i} x_j),$$

where $w_{j,i}$ is the weight from the jth neuron in the hidden layer to the ith neuron in the output layer, x_j is the activation value of the jth neuron in the hidden layer, Hid is a set consisting of all the hidden layer's neurons, and σ is the sigmoid function.

The modulatory activations are defined as

$$M_i = \Sigma_{j \in Mod} w^{\dagger}_{i,j} x^{\dagger}_j,$$

where $w^{\dagger}_{i,j}$ is the modulatory weight from the ith neuron in the output layer to the jth neuron in the modulatory layer, $x^{\dagger}_j$ is the input to the jth modulatory neuron and corresponds to the error computed at the jth output layer's neuron, and Mod is a set consisting of all the modulatory layer's neurons.

The weights between the hidden and output layer are updated according to

$$\Delta w_{j,i} = \sigma(M_i / n_{j,i}) \cdot \delta_{j,i},$$

where $n_{j,i}$ is a scaling parameter that is tuned while training, $\delta_{j,i}$ is the plasticity term, j corresponds to the jth neuron in the hidden layer, and i corresponds to the ith neuron in the output layer.

The plasticity term is defined as

$$\delta_{j,i} = \eta_{j,i} \cdot (A x_j x_i + B(x_i - x_j) + C),$$

where $\eta_{j,i}$ is an adaptive learning parameter that is updated while training, x_j is the output of the jth neuron (which is input to the ith neuron), x_i is the output of ith neuron, and the following three tunable parameters: the correlation term A, the difference term B, and the constant term C.

The adaptive learning parameter is defined as

$$\eta_{j,i} = \eta_{in} \frac{e_i}{x_i},$$

where η_{in} is the initial learning rate, e_i is the error at the ith output neuron, and x_i is the activation value of the ith output neuron.

The modulatory layer's weights are updated according to

$$\Delta w^{\dagger}_{j,i} = \eta^{\dagger}_{j,i}(scale),$$

where $\eta^{\dagger}_{j,i}$ is an adaptive learning parameter that differs to $\eta_{j,i}$ only in that e_i and x_i switch positions and $scale$ is a tunable magnitude parameter.

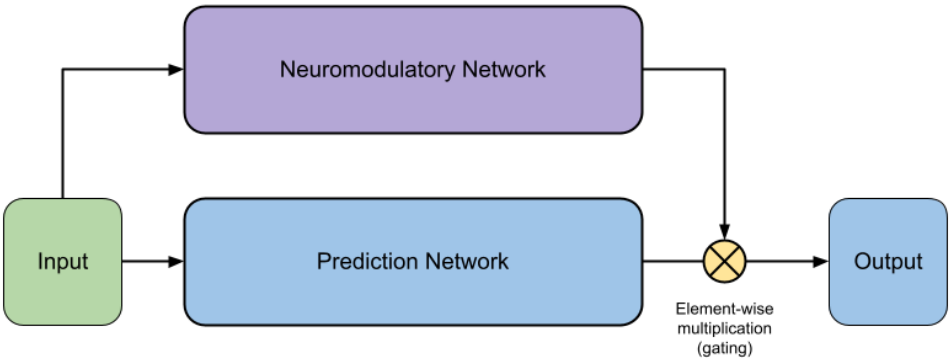


FIGURE 2.1: A general overview of ANML's architecture [6].

Incorporating neuromodulation with backpropagation

While the previous neuromodulation papers have focused on modifying the learning rates of parameters within a network, other approaches focus on integrating it with backpropagation. **A** **N**euromodulated **M**eta-**L**earning algorithm (ANML) utilises an architecture that consists of a prediction network and a neuromodulatory network [6] (see Figure 2.1 for an overview of the architecture). Both networks utilise the same input and can be structured as the user pleases. The general theme of the architecture is that the neuromodulatory network produces a matrix that, via element-wise multiplication, gates the prediction network at a user-designated point or points, introducing context-dependent gating and selective plasticity; it is inspired by neuromodulatory processes in the brain. Through gating, the neuromodulatory network has control of not only the forward propagation of the prediction network but the backpropagation of the error through it as well (i.e., selective plasticity). The architecture can be meta-learned like is done in [6], or it could be trained in the traditional sense in a non-continual learning domain.

Another paper that entwines neuromodulation with backpropagation is [58]. They introduce Backpropamine, which allows neuromodulated plasticity to be differentiable and trained with the backpropagation of the error signal produced by a network. The introduced approach builds upon differentiable Hebbian plasticity. Hebbian plasticity allows for the plasticity of each connection between neurons to be optimised via backpropagation. Each connection consists of two components: a fixed component that is a typical weight update and a plastic component that grows and decays conditioned on recent activity. The output of the ith neuron at time t is

$$o_i(t) = \sigma\{\Sigma_{j \in Inp}(w_{j,i} + \alpha_{j,i} Hebb_{j,i}(t)) o_j(t-1)\},$$

where σ is a non-linear function, $w_{j,i}$ is the weight between the jth and ith neuron, $o_j(t)$ is the output of the jth neuron at time t, $\alpha_{j,i}$ is the plastic coefficient optimised by the network via gradient descent, Inp is the set of neurons that are input to neuron i, and $Hebb_{j,i}(t)$ is the Hebbian trace, the plastic component of the connection.

The Hebbian trace is defined as

$$Hebb_{j,i}(t+1) = Clip(Hebb_{j,i}(t) + \eta o_j(t-1) o_i(t)),$$

where $Clip$ is a function that constrains the input to values in [-1,1] and η is the intra-life learning rate (i.e., the learning rate of the plastic connection). The intra-life learning rate determines how quickly new information is incorporated into the

Hebbian trace.

A simple implementation of neuromodulation with the Hebbian trace is

$$Hebb_{j,i}(t+1) = Clip(Hebb_{j,i}(t) + M(t)o_j(t-1)o_i(t)),$$

where the only difference is that η is replaced with $M(t)$, a single scalar output of the network that is either used as-is or is further modified by a meta-learned vector of weights (one for each connection between neurons).

Additionally, a more complex neuromodulation rule is introduced and is inspired by the "short-term retroactive effects of neuromodulatory dopamine on Hebbian plasticity in animal brains" [58]. The Hebbian trace is modified with the addition of an eligibility trace $E_{j,i}(t)$ for the connection between the jth and ith neurons. The eligibility trace keeps an account of which neurons (synapses) have contributed recently.

The updated Hebbian trace is defined as

$$Hebb_{j,i}(t+1) = Clip(Hebb_{j,i}(t) + M(t)E_{j,i}(t)),$$

where the eligibility trace is formulated as

$$E_{j,i}(t+1) = (1-\eta)E_{j,i}(t) + \eta o_j(t-1)o_i(t),$$

where η is a trainable decay factor and $E_{j,i}$ is an exponential moving average of the product between the ith neuron's output at time t and the jth neuron's output at the previous time step $t-1$. A dopamine signal $M(t)$ modulates the eligibility trace.

NMN is architecture introduced in [91] that consists of a *neuromodulatory* network and a *main* network. The main network is a typical ANN of potentially any structure that differs only in that all activation functions are replaced with a neuromodulatory version, generated by the neuromodulatory network. The modulatory network takes as input some context c and generates a single neuromodulatory signal $z \in \mathbb{R}^k$ that is used for all activations in the main network (k is a free parameter); it modifies the slope and bias of the main network through its activations. The neuromodulatory activations are represented by

$$\sigma(z^T(xw_s + w_b)),$$

where $w_s \in \mathbb{R}^k$ and $w_b \in \mathbb{R}^k$ are two vectors of the activation function that are trainable, representing the scale factor which modifies the slope and the offset of the activation function, respectively; σ is a non-linear activation function that maps a real number to another real number; $x \in \mathbb{R}$.

Additionally, while not explicitly labelled as neuromodulation in their paper — it does fit the criteria for neuromodulation — a Text Saliency Model is utilised and produces attention scores, which are utilised by a separate network's (the Task Model's) attention layer [83]. The attention scores produced by the Text Saliency Model can be thought of as providing additional context to the Task Model, dynamically controlling the attention layer and its produced output. The high-level description of the architecture is not too dissimilar to ANML's architecture [6], both consist of two models that share the same input, with one influencing the forward propagation of another by proving additional context, although the Text Saliency Model is explicitly trained to perform a specific task, while ANML's architecture is not.

2.4 Transformers

The Transformer [90] is an architecture constructed to overcome the bottleneck introduced by the sequential nature of recurrent sequence-to-sequence architectures that utilise an attention mechanism such as RNNsearch [3]. The Transformer eschews recurrence, allowing for more parallelisation to occur and introduces a foundational architecture that has been built upon in the QA domain [10, 23, 50, 65, 66]. This section gives a detailed description of the vanilla Transformer [90] and an overview of two descendant of it: Bidirectional Encoder Representations from Transformers (BERT) [23] and Generative Pre-trained Transformer (GPT) [10, 65, 66].

2.4.1 Attention

In machine learning models an *attention mechanism* is defined as a process that selectively attends to either the input or certain activations throughout the forward propagation of an ANN. It is inspired by attention in humans, where we selectively attend to a subset of the external stimuli available to us [31]. By attending we mean turning on or off certain activations or parts of the input so that the model can focus on a subset of them. The goal with such a mechanism is that if the model learns to attend effectively, then the model as a whole will be more efficient as it only focuses on what is relevant.

We will cover two attention mechanisms utilised in the NLP domain, but first, we will provide further definitions of attention mechanisms. The first is **self-attention**: an attention mechanism in which each position of the input sequence attends to all other positions in the same sequence [13]; the alternative to self-attention is attending to a different sequence altogether. The second is **soft-attention**: an attention mechanism in which a probability distribution is induced across all attending to tokens, i.e., a soft alignment over the sequence [99]. This differs from **hard-attention** whereby instead of softly aligning over the input, each item that is attended to is either fully included or excluded, no in-between [99]. All attention mechanisms that we cover fit into the category of self-attention and soft-attention.

The first attention mechanism which we will cover is *additive attention*, which was designed to fit into the encoder-decoder framework of sequence-to-sequence models [3]. Given a set of vectors $\{h_1, h_2, ..., h_N\}$ of an encoder — each of dimension d — that we would like to attend over, we compress it into a single vector c_i, the context vector of the ith decoder position; the vectors are alternatively referred to as annotations. It is defined as

$$c_i = \Sigma_{j=1}^{N} \alpha_{i,j} h_j,$$

where h_j is the j^{th} annotation of the encoder and α_i represents a probability distribution with $\alpha_{i,j}$ representing the probability assigned to the j^{th} annotation. Each annotation is weighted by $\alpha_{i,j}$ by multiplying each element in the annotation by $\alpha_{i,j}$; all weighted annotations are added together via element-wise addition.

The probability value of the j^{th} annotation for a fixed decoder position i is

$$\alpha_{i,j} = \frac{exp^{e_{i,j}}}{\Sigma_{k=1}^{N} exp^{e_{i,k}}},$$

where $e_{i,j} \in \mathbb{R}$ is the associated energy between the ith decoder position and jth annotation; exp represents Euler's number. Put simply, this is the softmax function applied to all associated energies; in this instance, it returns the probability for the jth annotation.

The associated energies are calculated via

$$e_{i,j} = a(s_{i-1}, h_j),$$

where $s_{i-1} \in \mathbb{R}^d$ is the previous decoder position's vector (i.e., internal state), $h_j \in \mathbb{R}^d$ is the *jth* annotation of the encoder, and a is a function that computes the alignment score.

One way to calculate the alignment score is

$$a(s_{i-1}, h_j) = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right),$$

where $W_a \in \mathbb{R}^{d\times d}$, $U_a \in \mathbb{R}^{d\times d}$, $s_{i-1} \in \mathbb{R}^{d\times 1}$, $h_j \in \mathbb{R}^{d\times 1}$, and $v_a \in \mathbb{R}^{d\times 1}$. For ease of explanation the encoder and decoder's hidden states have been converted from vectors into matrices of dimension $d \times 1$. Dot-product multiplication is performed between W_a and the decoder's previous hidden state s_{i-1}, and between U_a and the j^{th} annotation h_j. Element-wise addition is performed between the two resulting matrices, with the tanh activation function being applied directly after; the resulting matrix is of dimension $d \times 1$ and is denoted by Z. Dot-product multiplication is performed between the transpose of v_a and Z, producing a single real number which is taken as the alignment score. The name "additive attention" comes from the fact that addition is the operation used to combine decoder's previous hidden state and the *jth* annotation.

The second attention mechanism is *dot-product attention*, which is similar to additive attention, but replaces the addition operation with a dot-product operation [90]. Dot-product attention consists of a query, key, and value; in the context of information retrieval, the role of the query is to be matched against keys and their associated values. Given a query matrix Q, key matrix K, and value matrix V, it is defined as

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

where $Q \in \mathbb{R}^{N_q\times d_q}$, $K \in \mathbb{R}^{N_k\times d_k}$, and $V_k \in \mathbb{R}^{N_v\times d_v}$ are the input matrices, representing the query, key, and value, respectively; N_q, N_k, and N_v represent the sequence length for the query, key, and value, respectively; d_q, d_k, and d_v represent the dimension of the query, key, and value, respectively. The query and the transposed key matrices are multiplied together via dot-product multiplication, then scaled by the square root of the dimension of the key matrix $\sqrt{d_k}$, resulting in a new matrix of dimension $N_q \times N_k$. The softmax activation function is applied to the resulting matrix, inducing a probability distribution across rows; i.e., the sequence represented by the query attends to that of the key. This is referred to as the attention matrix, which is then multiplied with the value matrix via dot-product multiplication, generating a new matrix of dimension $N_q \times d_v$, concluding dot-product attention. Note that d_q and d_k, and N_k and N_v must be equal.

In both additive and dot-product attention the attention matrix c is calculated, where the sequence corresponding with the rows attends to the sequence corresponding with the columns. For additive attention to generate the context matrix c it needs to process the decoder's input n_{dec} times, and the encoder's input n_{enc} times: $n_{dec} \times n_{enc}$ in total. While similar to dot-product attention, the difference is that dot-product attention processes everything at once, not sequentially, allowing efficient matrix multiplication to cover the whole process. For additive attention this occurs sequentially along the decoder axis, meaning that the efficient matrix multiplication code is split up n_{dec} times.

To be definitively clear, it is not additive attention alone that causes the issue per se, it is the fact that it is processed sequentially. We could replace the alignment score function a with the following equation

$$a(s_{i-1}, h_j) = f(s_{i-1})g(h_j)^T,$$

which integrates the query, key, and value terminology in the alignment score calculation and utilises the dot-product operation. The same issues still hold, i.e., the sequential nature in which it is entwined.

2.4.2 Vanilla Transformer

The Transformer is an architecture that was introduced to eliminate the reliance on recurrent architectures, by instead relying entirely on attention [90]. This is achieved by the newly introduced *dot-product attention*, which differs from additive attention in the scaling parameter and how the alignment score is computed. The main advantage of this new attention mechanism is the increased efficiency by allowing the attention matrix to be calculated — across the whole sequence — in a single matrix multiplication, meaning that efficient matrix multiplication code can be utilised.

Figure 2.2 showcases the Transformer architecture; it consists of an encoder and decoder. The encoder produces an encoded representation of the input sequence and the decoder utilises the encoded representation and generates an answer to a task sequentially. During training, teacher forcing at each position is utilised by the decoder, allowing the entire target answer to be processed at once, significantly speeding up training over its recurrent counterparts; however, note that during evaluation the answer will need to be processed sequentially, eliminating the advantage of dot-product attention in the decoder. Both the encoder and decoder have multiple layers, of which each contains multiple blocks. Each block consists of a residual connection, layer normalization layer, and either a point-wise feed-forward module or a multi-head attention module.

Given that the model eschews recurrence, the word embedded input to the model has no positional information about a token's relation to other tokens in the input sequence. Therefore, positional information about the input sequence needs to be added to the input. *Fixed absolute position embeddings* are utilised to fill this void. They are fixed in the sense that they are not learned but constant and absolute in the sense that it adds positional information to a token about its position in the sequence as a whole, not relative to other tokens. It is defined as

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}), \end{aligned} \tag{2.1}$$

where pos is a position in the input sequence, i is the dimension index, and d_{model} is the dimension of the Transformer and the word embeddings. Each dimension i alternates between a sine and cosine function for even and odd dimension indices, respectively, with the wavelengths forming a geometric progression from 2π to $10000 \times 2\pi$. The resulting matrix from the embedding will be of the same dimension as the input sequence embeddings: $N \times d_{model}$, where N is the length of the input sequence. Alternatively, a learned position embedding matrix of the same dimensions can be used in place of the fixed absolute position embedding.

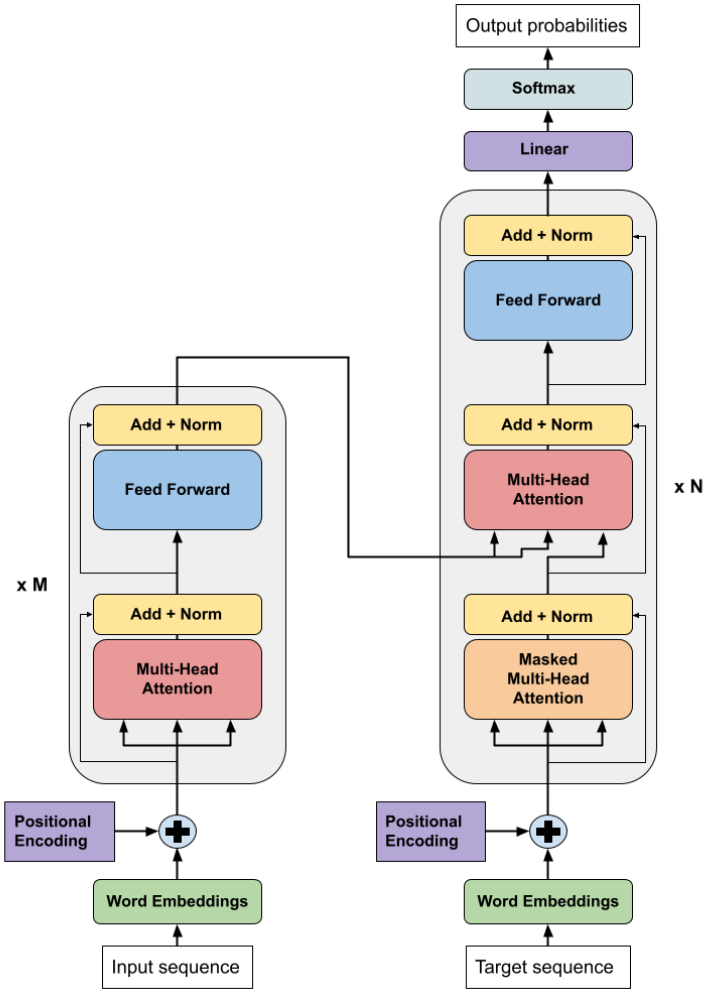


FIGURE 2.2: The Transformer architecture introduced in [90]. The encoder consists of M layers and the decoder consists of N layers.

Each block of both the encoder and decoder consists of a layer normalization layer and residual connection. Layer normalization fulfils the purpose of normalizing neurons passed as input, reducing the training time, especially for larger neural networks [2]. This is because the changing outputs in one layer often cause highly correlated changes to the input of the next layer. Normalizing via fixing the mean and variance of the input curtails the issues introduced by the correlated changes, helping reduce training time. It is defined as

$$LayerNorm(X) = \theta \frac{X - \mu}{\sigma} \delta, \tag{2.2}$$

where μ is the mean, σ is the standard deviation, and θ and δ are the gain and bias parameters, respectively. The bias and gain parameters are of the same dimension as the input matrix X. The mean is defined as

$$\mu = \frac{1}{H} \Sigma_{i=1}^{H} a_i \tag{2.3}$$

and the standard deviation is defined as

$$\sigma = \sqrt{\frac{1}{H}\Sigma_{i=1}^{H}(a_i - \mu)^2}, \tag{2.4}$$

where in both instances H is the number of hidden units and a_i represents the summed inputs to the ith neuron.

A residual connection fulfils the purpose of helping reduce the issue of vanishing gradients, allowing for faster training times, larger networks to be trained and better performance [32]. It is defined as

$$y = \mathbb{F}(x, \{W_i\}) + x, \tag{2.5}$$

where $\mathbb{F}(x, \{W_i\})$ is the residual mapping to be learned, x is the input matrix, and W_i is the weights applied to the input matrix. The output y is defined to be the element-wise addition between the input matrix x and the residual mapping.

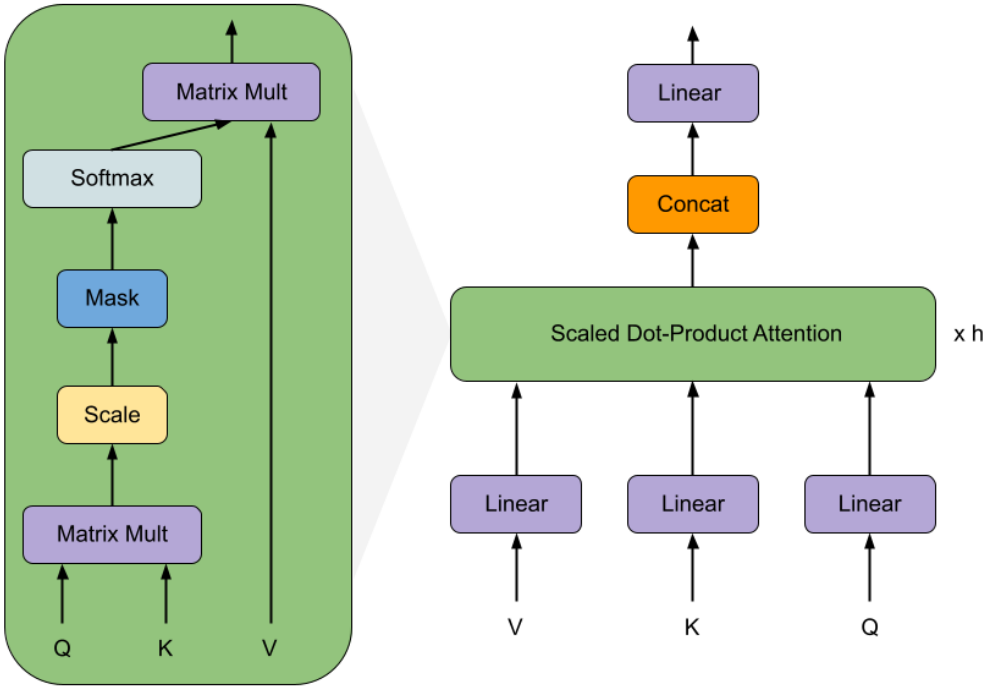


FIGURE 2.3: Scaled dot-product attention is shown on the left and multi-head attention is shown on the right.

Note that for the rest of this section d_{model}, d_{ff}, d_q, d_k, d_v, N_q, N_k and N_v refer to the dimension of the Transformer, dimension of the point-wise feed-forward network, dimension of the query matrix, dimension of the key matrix, dimension of the value matrix, length of the query's associated sequence, length of the keys' associated sequence, and length of the value's associated sequence, respectively. In general terms, the sequence length is denoted by N; the sequence length for the encoder and decoder is denoted by N_{enc} and N_{dec}, respectively. In practice d_q, d_k, and d_v are all equal to d_{model} in the Transformer. Additionally, when we reference dot-product attention, we are referring to scaled dot-product attention as shown in Figure 2.3.

A block also consists of either a multi-head attention module or a point-wise feed-forward module. The multi-head attention module takes as input a query, key,

and value matrix. It is defined as

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,\ldots,head_h)W^O \\ \text{where } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V), \end{aligned} \tag{2.6}$$

where the input matrices are $Q \in \mathbb{R}^{N_q \times d_q}$, $K \in \mathbb{R}^{N_k \times d_k}$, and $V \in \mathbb{R}^{N_v \times d_v}$, representing the query, key, and value, respectively. The query, key, and value matrices are linearly projected via dot-product multiplication with $W_i^Q \in \mathbb{R}^{d_{model} \times d_{model}}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_{model}}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_{model}}$, respectively, resulting in a new matrices of the same dimensions. The linearly projected matrices are split into h evenly sized subsets, each of dimension $N_q \times (d_q/h)$, $N_k \times (d_k/h)$, and $N_v \times (d_v/h)$ for the linearly projected query, key, and value, respectively. The ith subsets (represented by QW_i^Q, KW_i^K, and VW_i^V) are input to a dot-product attention module, where the output is a new matrix of dimension $N_q \times (d_v/h)$; denote this by $head_i$. Note that h must be a factor of d_q, d_k, and d_v. To conclude multi-head attention, all heads are concatenated together to form a new matrix of dimension $\mathbb{R}^{N_q \times d_v}$, which is then multiplied with $W^O \in \mathbb{R}^{d_v \times d_{model}}$ via dot-product multiplication, resulting in a new matrix of dimension $\mathbb{R}^{N_q \times d_{model}}$, concluding multi-head attention.

The dot-product attention module's focal point is an attention matrix that induces a probability distribution across rows (i.e., each row sums to one) with each column representing a token to attend to. It is defined as

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.7}$$

where the input matrices are $Q \in \mathbb{R}^{N_q \times d_q}$, $K \in \mathbb{R}^{N_k \times d_k}$, and $V \in \mathbb{R}^{N_v \times d_v}$, representing the query, key, and value, respectively. The query matrix is multiplied with the transposed key matrix via dot-product multiplication to produce a new matrix of dimension $N_q \times N_k$, which is then scaled by the square root of the scaling parameter $\sqrt{d_k}$. The softmax activation function is applied across rows to the scaled matrix to produce an attention matrix $A \in \mathbb{R}^{N_q \times N_k}$. The attention matrix induces a probability distribution across rows, i.e., the sequence represented by the query attends to that of the key. Dot-product multiplication is performed between the attention matrix and the value matrix V, producing a new matrix of dimension $N_q \times d_v$, concluding dot-product attention. Note that d_q, d_k, and d_v are always equal in the Transformer; N_k and N_v must always be equal; for self-attention, N_q and N_k must be equal. In practice, some of the tokens in the attention matrix are masked meaning that they cannot be attended to by some tokens (see Figure 2.4 for an overview of masking).

The point-wise feed-forward network module consists of two fully connected layers, one directly after the other in a typical neural network. It is defined as

$$\begin{aligned} a_1 &= max(0, XW_1) \\ a_2 &= a_1 W_2 \\ FFN(X) &= a_2, \end{aligned} \tag{2.8}$$

where the input is a matrix $X \in \mathbb{R}^{N \times d_{model}}$, and $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ represent the weight matrices of the first and second fully connected layers, respectively. The matrix X is multiplied with W_1 by dot-product multiplication and has the ReLU activation function applied directly after, constructing a new matrix

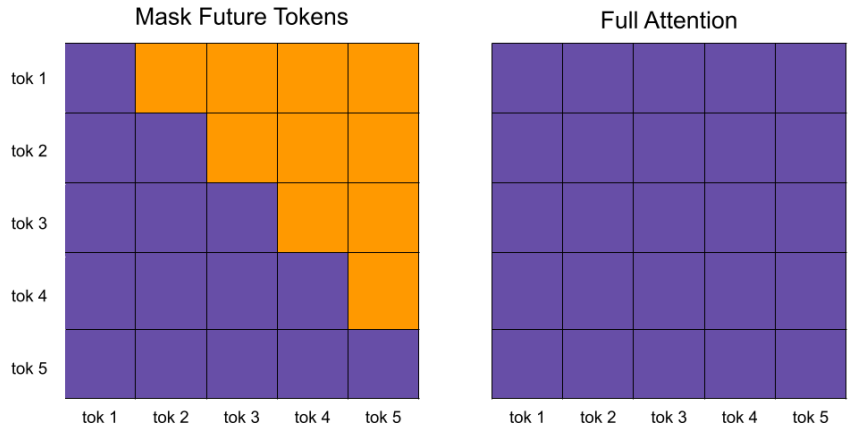


FIGURE 2.4: Types of masking in the attention matrix: purple indicates that attending to that location is allowed, while orange indicates that it is not allowed (i.e., they are masked). The tokens on the y-axis attend to positions on the x-axis; all attention matrices shown are performing self-attention. In uni-directional models all future tokens are masked; it is displayed in the "Mask Future Tokens" matrix. Bi-directional models utilise full attention, where no positions are masked; it is depicted in the "Full Attention" matrix.

$a_1 \in \mathbb{R}^{N \times d_{ff}}$. Dot-product multiplication is performed between a_1 and W_2, producing a new matrix $a_2 \in \mathbb{R}^{N \times d_{model}}$, concluding the point-wise feed-forward network.

In the Transformer there exists two types of blocks, the *point-wise feed-forward network block* and *multi-head attention block*. The point-wise feed-forward network block is defined as

$$\begin{aligned} x_1 &= FFN(X) \\ x_2 &= LayerNorm(x_1 + X) \\ pwffn_{\text{block}}(X) &= x_2, \end{aligned} \tag{2.9}$$

where the input matrix $X \in \mathbb{R}^{N \times d_{model}}$ is input to the point-wise feed forward module FNN (2.8), producing a new matrix x_1 of the same dimension. Layer normalization [2] is applied to the element-wise addition of x_1 and X (i.e., a residual connection [32]), resulting in a new matrix matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$, which is taken as the output of the block.

The multi-head attention block is defined as

$$\begin{aligned} x_1 &= MultiHead(Q, K, V) \\ x_2 &= LayerNorm(x_1 + Q) \\ mha_{\text{block}}(Q, K, V) &= x_2, \end{aligned} \tag{2.10}$$

where the input matrices are $Q \in \mathbb{R}^{N_q \times d_{model}}$, $K \in \mathbb{R}^{N_k \times d_{model}}$, and $V \in \mathbb{R}^{N_v \times d_{model}}$, representing the query, key, and value matrices, respectively. All three are passed as input to the multi-head attention module (2.6), which produces a new matrix $x_1 \in \mathbb{R}^{N_q \times d_{model}}$. Similar to that of the point-wise feed-forward block, a layer normalization layer [2] is utilised with the input being the element-wise addition between x_1 and Q (i.e., a residual connection [32]), producing a new matrix $x_2 \in \mathbb{R}^{N_q \times d_{model}}$ that is taken as the output of the block.

Let the following be a sample QA task where "Who is the greatest Laker of all time?" is the encoder's input and "Kobe Bryant" is the decoder's input. Table 2.2 illustrates a sample vocabulary V for both the encoder and decoder's input.

TABLE 2.2: Sample vocabulary that maps words to ids.

Vocabulary	ID
<unk>	0
who	1
is	2
the	3
greatest	4
laker	5
of	6
all	7
kobe	8
time	9
?	10
...	...
bryant	24

Firstly, we need to process the input and convert it to a Transformer readable format. This is achieved through tokenization, where the input is processed and mapped to a unique id. The processed input could look like $X_{\text{enc-processed}} = \{\text{who, is, the, greatest, laker, of, all, time, ?}\}$ and $X_{\text{dec-processed}} = \{\text{kobe, bryant}\}$ depending on the processing strategy. Let $f : V \to \{0 \leq \mathbb{Z} \leq |V|\}$ be the mapping from the vocabulary V to $|V| + 1$ integers, where $|V|$ is the size of the vocabulary. Note that the reason for an additional integer is to represent tokens not in the vocabulary: unknown tokens. Applying f to the processed inputs results in the following matrices: $X_{\text{enc-tokenized}} = \{1, 2, 3, 4, 5, 6, 7, 9, 10\}$ and $X_{\text{dec-tokenized}} = \{8, 24\}$ for the encoder and decoder's input, respectively.

The input needs to be converted to a vector so it can be processed by the Transformer. Let $g : \{0 \leq \mathbb{Z} \leq |V|\} \to d_{model}$ be the mapping from of an integer to a vector of dimension d_{model}. After applying $X_{\text{enc-tokenized}}$ and $X_{\text{dec-tokenized}}$ to g for every token in the sequence we end up with the matrices $X_{enc} \in \mathbb{R}^{N_{enc} \times d_{model}}$ and $X_{dec} \in \mathbb{R}^{N_{dec} \times d_{model}}$, where N_{enc} and N_{dec} represent the length of the input sequence for both the encoder and decoder, respectively. Both the encoder and decoder's input will have a positional encodings applied to them as defined previously. Let $X_{\text{enc-pos}}$ and $X_{\text{dec-pos}}$ be the resulting matrices after positional encoding is applied to the encoder and decoder's input, respectively.

The encoder of the Transformer consists of L_{enc} layers stacked on top of one another, with the *ith* layer being defined as

$$\begin{aligned} x_1 &= mha_{\text{block}}(X, X, X) \\ x_2 &= pwffn_{\text{block}}(x_1) \\ Encoder^i_{\text{layer}}(X) &= x_2, \end{aligned} \tag{2.11}$$

where the input matrix $X \in \mathbb{R}^{N_{enc} \times d_{model}}$ plays the role of the query, key, and value in the multi-head attention block (2.10), i.e., self-attention is performed. A new matrix $x_1 \in \mathbb{R}^{N_{enc} \times d_{model}}$ is constructed. The point-wise feed-forward network block (2.9)

takes as input x_1 and generates a new matrix $x_2 \in \mathbb{R}^{N_{enc} \times d_{model}}$, which is taken as the output of the *ith* encoder layer.

The encoder as a whole is defined as

$$\begin{aligned} O^1_{enc} &= Encoder^1_{\text{layer}}(X_0) \\ O^i_{enc} &= Encoder^i_{\text{layer}}(O^{i-1}_{enc}) \; \forall \; \{1 < i \leq L_{enc}\} \\ Transformer_{\text{encoder}}(X_0) &= O^{L_{enc}}_{enc}, \end{aligned} \tag{2.12}$$

where the input matrix $X_0 \in \mathbb{R}^{N_{enc} \times d_{model}}$ is the input to the first layer of the encoder (2.11). A new matrix $O^1_{enc} \in \mathbb{R}^{N_{enc} \times d_{model}}$ is produced and is input to the next encoder layer. This process repeats until the L^{th}_{enc} encoder layer generates a new matrix $O^{L_{enc}}_{enc} \in \mathbb{R}^{N_{enc} \times d_{model}}$, which is taken as the output of the encoder.

The decoder of the Transformer consists of L_{dec} layers stacked on top of one another. The *ith* layer is defined as

$$\begin{aligned} x_1 &= mha_{\text{block}}(X, X, X) \\ x_2 &= mha_{\text{block}}(x_1, O^{L_{enc}}_{enc}, O^{L_{enc}}_{enc}) \\ x_3 &= pwffn_{\text{block}}(x_2) \\ Decoder^i_{\text{layer}}(X, O^{L_{enc}}_{enc}) &= x_3, \end{aligned} \tag{2.13}$$

where the input matrices are $X \in \mathbb{R}^{N_{dec} \times d_{model}}$ and $O^{L_{enc}}_{enc} \in \mathbb{R}^{N_{enc} \times d_{model}}$, representing the decoder's input and the encoded representation of the encoder's input, respectively. The matrix X is utilized as the query, key, and value of the first multi-head attention block (2.10), i.e., self-attention is performed. A new matrix $x_1 \in \mathbb{R}^{N_{dec} \times d_{model}}$ is constructed. A second multi-head attention block (2.10) utilizes the matrix x_1 as the query, and the encoded representation $O^{L_{enc}}_{enc}$ as the key and value, outputting a new matrix $x_2 \in \mathbb{R}^{N_{dec} \times d_{model}}$. Lastly, a point-wise feed-forward network block (2.9) takes the matrix x_2 as input and assembles a new matrix $x_3 \in \mathbb{R}^{N_{dec} \times d_{model}}$, which is the output of the *ith* decoder layer.

The decoder as a whole is defined as

$$\begin{aligned} O^1_{dec} &= Decoder^1_{\text{layer}}(X_0, O^{L_{enc}}_{enc}) \\ O^i_{dec} &= Decoder^i_{\text{layer}}(O^{i-1}_{dec}, O^{L_{enc}}_{enc}) \; \forall \; \{1 < i \leq L_{dec}\} \\ Transformer_{\text{decoder}}(X_0, O^{L_{enc}}_{enc}) &= O^{L_{dec}}_{dec}, \end{aligned} \tag{2.14}$$

where the input matrices are $X_0 \in \mathbb{R}^{N_{dec} \times d_{model}}$ and $O^{L_{enc}}_{enc} \in \mathbb{R}^{N_{enc} \times d_{model}}$, representing the input sequence of the decoder and the encoded representation of the encoder's input, respectively. They both are input to the first decoder layer (2.13), which produces a new matrix $O^1_{dec} \in \mathbb{R}^{N_{dec} \times d_{model}}$. For all subsequent decoder layers the input is the output of the previous decoder layer $O^{i-1}_{dec} \in \mathbb{R}^{N_{dec} \times d_{model}}$ and the same encoded representation $O^{L_{enc}}_{enc}$. This process repeats until the L^{th}_{dec} decoder layer generates a new matrix $O^{L_{dec}}_{dec} \in \mathbb{R}^{N_{dec} \times d_{model}}$, which is taken as the output of the decoder.

The Transformer as a whole consists of an encoder and decoder as described previously. It is defined as

$$\begin{aligned} O_{enc}^{L_{enc}} &= Transformer_{\text{encoder}}(X_{encoder}) \\ O_{dec}^{L_{dec}} &= Transformer_{\text{decoder}}(X_{decoder}, O_{enc}^{L_{enc}}) \\ O_{logits} &= O_{dec}^{L_{dec}} W_{output} \\ O_{prob} &= softmax(O_{logits}) \\ Transformer(X_{encoder}, X_{decoder}) &= O_{prob}, \end{aligned} \tag{2.15}$$

where the input matrices are $X_{encoder} \in \mathbb{R}^{N_{enc} \times d_{model}}$ and $X_{decoder} \in \mathbb{R}^{N_{dec} \times d_{model}}$, representing the encoder and decoder's input, respectively. Utilising our previously constructed example the encoder and decoder's input would be $X_{\text{enc-pos}}$ and $X_{\text{dec-pos}}$, respectively. The encoder and decoder is traversed as described in (2.12) and (2.14), respectively; producing new matrices $O_{enc}^{L_{enc}} \in \mathbb{R}^{N_{enc} \times d_{model}}$ and $O_{dec}^{L_{dec}} \in \mathbb{R}^{N_{dec} \times d_{model}}$, respectively. The output matrix of the decoder $O_{dec}^{L_{dec}} \in \mathbb{R}^{N_{dec} \times d_{model}}$ is dot-product multiplied with $W_{output} \in \mathbb{R}^{d_{model} \times |V_d|}$, producing a new matrix $O_{logits} \in \mathbb{R}^{N_{dec} \times |V_d|}$. It is then passed through a softmax activation function, inducing a probability distribution over the target vocabulary for each position in the target sequence; it produces a new matrix $O_{prob} \in \mathbb{R}^{N_{dec} \times |V_d|}$, where V_d is the vocabulary of the decoder and $|V_d|$ is the size of the vocabulary. For each position in the target sequence, the maximum probability and the token that it represents is chosen as the prediction, concluding the Transformer.

2.4.3 BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) is a bi-directional masked language model [23]. BERT is practically identical to the encoder in the vanilla Transformer [90] with minor differences: the GELU activation function [34] replaces the ReLU activation function and the input embedding process is modified to include segment embeddings.

The input to BERT is a sequence of text with a classification token $[CLS]$ appended to the beginning of the sequence and a separator token $[SEP]$ that splits the input into segments.

Example. *[CLS] Shaquille O'Neal is the most dominant basketball player ever [SEP] He has broken the backboard many times [SEP]*

The input sequence is converted to a machine readable format by converting all tokens into their associated WordPiece embeddings [98] via a tokenizer that processes the input. As with the vanilla Transformer, positional information is missing, hence, absolute position embeddings are utilised; with the addition of segments via the $[SEP]$ token, each segment has a unique embedding vector associated with it, which is added to every token's embedding vector in the associated segment.

The output of BERT is no different to that in the vanilla Transformer's encoder except for an additional fully connected layer being inserted after the last encoder layer. BERT can perform various tasks: classification via the $[CLS]$ token, tasks that require predictions at each position in the input (e.g., parts of speech tagging), and text generation via the last token in the sequence.

BERT is pre-trained on a masked language modelling and next sentence prediction task. Masked language modelling refers to a task where tokens in a segment of

text are masked out, with the goal being to predict the original tokens in the masked positions given the context, i.e., the surrounding tokens. Of the tokens in the sequence, 15% are randomly selected to be masked. Of the masked positions, 80% of the time they are replaced with the $[MASK]$ token, 10% of the time with a random token, and 10% of the time they are unchanged. Next sentence prediction refers to a task where two segments of text, separated by the $[SEP]$ token, are predicted by the $[CLS]$ token position: true if the second segment follows the first and false otherwise.

The pre-trained BERT model can be used as a starting point to fine-tune further on specific tasks, utilising the features learned about language as a starting point and building upon them. At the time of the paper's publication, it achieved SOTA performance in many NLP tasks by fine-tuning on said task [23].

2.4.4 GPT

The **G**enerative **P**re-trained **T**ransformer (GPT) family of architectures [10, 65, 66] involves training a decoder only Transformer on a large chunk of text in an auto-regressive language modelling task. Unlike BERT, GPT models cannot see future tokens in the attention component (i.e., it is uni-directional), simulating the situation the model would encounter in evaluation mode, i.e., the generation of the tokens one by one. Here, we will focus on the GPT architecture present in GPT-2 [66].

Compared to the original Transformer decoder [90], there are some modifications to the architecture: the omission of the second multi-head attention block in all layers (that would otherwise take as input the encoded representation if it existed); in the point-wise feed-forward network, the ReLU activation function is replaced with the GELU activation function [34]; in each block, the layer normalization layer is moved to the beginning.

The input to GPT is a sequence of text with additional special tokens that perform specific tasks. The following example showcases such special tokens: <*s*> and </*s*>, representing the start and end tokens, respectively.

Example. <*s*> *Shaquille O'Neal has broken the backboard many times* </*s*>

The input sequence is converted to a machine readable format via a Byte Pair Encoding (BPE) [78]. In practice, it is wrapped up in a tokenizer that processes the text in its entirety, converting the sequence into its associated embedding matrix via BPE. Positional information is lacking in the embedding matrix, therefore, absolute position embeddings are utilised, as in the vanilla Transformer.

The output of GPT is identical to that of the Transformer decoder, including its utilisation of a fully connected layer with a softmax activation function to generate a prediction at every position in the sequence. With the utilisation of additional special tokens in the input, different fine-tuning tasks can be performed including many different types of QA datasets: MQA, span selection, answer generation, etc.

GPT is pre-trained on an auto-regressive language modelling task, where it generates a token one by one conditioned on the previously generated tokens acting as context. The probability of generating a sequence $y = (y_1, y_2, \ldots, y_N)$ is given by

$$\mathbb{P}(y) = \Pi_i^N \mathbb{P}(y_i | y_1, y_2, \ldots, y_{i-1}),$$

where the total probability $\mathbb{P}(y)$ is the product of the probabilities of each token in the sequence being generated given the context of all preceding tokens.

A pre-trained GPT model can be further fine-tuned on specific tasks such as QA, utilising the features learned about language as a starting point and building upon

them. Additionally, pre-training on a large amount of data and by a model with a large number of parameters, like GPT-3, allows for good performance on downstream tasks without any additional training in a zero-shot, one-shot, and few-shot setting [10]. A few-shot setting is when a few (more than one) examples of said downstream tasks — including the expected answer and its corresponding format — are provided with the current input; a one-shot setting only includes one example; a zero-shot setting includes no examples.

2.5 General question answering

Question answering (QA) involves the ability to perform inference and construct an answer to a question correctly if possible. It is an important sub-field in NLP since NLP tasks can be posed as questions [56]. Generalisation is an important, but often overlooked aspect in QA, with there existing a considerable gap in the current models between performance on a dataset and the task of the dataset [105], indicating that they do not generalise well within the same task, let alone QA in general. General question answering can be defined in many ways in the current literature: the performance of a single QA model with no further training on other datasets not seen during training and to any potential question that could be posed to it within reason [27, 41, 42]; in conjunction with the continual learning domain, the speed and efficiency it can adapt to new QA datasets or types of questions in general [105]. In this thesis, we focus on the former.

In this section, we will outline a variety of components of GQA in the literature: (1) relevant benchmarks that test generalisation; (2) the role multi-task learning, or the training procedure more generally, has on generalisation; (3) potential impacts the Transformer architecture has on generalisation; and (4) the input format, or more specifically, the text-to-text framework.

Benchmarks exist that test a single model's ability to perform on a wide variety of datasets [93, 94] and test their ability to generalise to out-of-domain datasets (i.e., datasets not seen during training) [27]. The GLUE [93] and SuperGLUE [94] benchmark both test a model's performance on a wide variety of natural language understanding tasks, with SuperGLUE consisting of more difficult language modelling tasks. The benchmark was originally constructed to promote models that acquire knowledge across tasks, i.e., general knowledge. The MRQA 2019 Shared Task [27] focuses on testing a model's ability to generalise to out-of-domain datasets, differing from the GLUE datasets where all datasets training sets are seen during training. Additionally, [41, 42] utilise their own benchmark with out-of-domain datasets.

One of the main contributing factors to the lack of generalisation in QA models is the training procedure [49, 87, 105], which determines what knowledge the model can extract. A multi-task learning objective like that in [49] implicitly encodes a regularisation effect where the model is unable to overfit to a single dataset. This notion is supported in [87], where they determine that a model overfits to individual datasets, while training on multiple datasets in a multi-task setting improves performance; additionally, they determine that more training data is beneficial to generalisation.

In [105], they test generalisation in terms of a model's ability to learn new tasks quickly (i.e., continual learning), with the fewer training instances the better, preferably zero. This definition brings in other aspects to the formula, such as the catastrophic forgetting issue that plagues the continual learning domain, meta-learning, and potential training curriculums. In their experiments, they find that a model

overfits to individual datasets and performs poorly on other datasets of the same task; it highlights the difference between learning in a dataset and a task. In the continual learning domain, they find that catastrophic forgetting runs rampant, forgetting previously acquired knowledge; however, they show that randomly sampling examples from a task (across multiple datasets) and training on them help alleviate the catastrophic forgetting phenomena, further supporting the notion that the training procedure is a major contributor to generalisation.

The architecture of the Transformer can have an impact on the generalisation capabilities of the model. For example, keeping a memory of past activations, like that in Transformer-XL [18] and the Compressive Transformer [67], allows for the model to generalise to longer sequence lengths. Given that out-of-domain datasets may require a context length greater than that encountered in training for good performance, the ability to generalise to longer sequence lengths is essential.

The text-to-text framework utilised in UnifiedQA [41, 42] and introduced in [68] converts QA datasets of different formats into a single unified format. Naturally, a model trained on one format will perform worse on a different format, hence, the text-to-text framework remedies the issue by converting all formats to a single unified format. UnifiedQA, combined with multi-format learning on QA datasets, shows the potential of unifying all formats, achieving SOTA results on a variety of datasets and good performance on out-of-domain datasets, not seen during training.

2.6 Datasets

In this thesis, we utilise language modelling datasets to pre-train a model, allowing it to acquire linguistic knowledge about language. We utilise multiple-choice, extractive, abstractive, and yes/no type QA datasets to test a model's proficiency in QA and GQA via the fine-tuning of the pre-trained model.

2.6.1 Language modelling

The metric used to measure a language model's performance is perplexity (PPL) [80]. Perplexity is a measurement of how well a model predicts a sample of text; the lower the perplexity score the better the model is. Perplexity is calculated by $e^{mean(loss)}$, where e is Euler's number and $mean(loss)$ is the average cross entropy loss over a corpus.

The **C**olossal **C**lean **C**rawled **C**orpus (C4) is an English corpus consisting of filtered text scraped from the web via Common Crawl [68]. The resulting data is around (750 GB); Hugging Face hosts a compressed variant of 305 GB that we use for pre-training in this thesis[1].

LAnguage **M**odelling **B**roadened to **A**ccount for **D**iscourse **A**spects (LAMBADA) is a dataset that tests a language model's understanding of broader context in natural text from novels [62]. The task is to predict the last word in the provided sequence of text. The training set consists of 2,662 novels and 203 million words; training is a typical language modelling task. The validation and test sets consist of 4,869 and 5,153 passages, respectively, each coming from disjoint novels distinct from those in the training set. The average number of tokens in each is 75.4 and 75 for the validation and test sets, respectively.

[1] https://huggingface.co/datasets/c4

The metrics used to obtain results is the exact-match accuracy of predicting the final world correctly; alternatively, the whole passage could be utilised in its entirety and have the perplexity score reported. We do the latter in this thesis.

WikiText is a language modelling dataset made up of Wikipedia articles. There exist two versions of the dataset: WikiText-2 and Wikitext-103, which correspond to the 2 million and 103 million word versions, respectively [57]. Both datasets share the same validation and test sets with the only difference being the vocabulary. Perplexity is the default metric used to measure a model's language model capabilities on WikiText.

The Penn Treebank (PTB) is an American English corpus of over 4.5 million words [53]. It has been pre-processed by [60], shrinking the corpus to a training set of 930,000 words, a validation set of 74,000 words, and a test set of 82,000 words. We utilise the pre-processed version and utilise perplexity as our metric to evaluate performance.

2.6.2 Multiple-choice

The metric to measure performance on MQA datasets is the number of correctly predicted questions divided by the total number of questions. More precisely, the exact-match accuracy of the correct label being generated.

Re**A**ding **C**omprehension Dataset from **E**xaminations (RACE) is a high quality, large reading comprehension dataset taken from English examinations for middle school (ages 12-15) and high school (ages 15-18) Chinese students [45]. The dataset consists of 27,933 passages and 97,687 questions, each with four answer options. The dataset is split into two subsets: RACE-m and RACE-h, representing the middle and high school subsets, respectively.

For the RACE-m subset, the training set consists of 6,409 passages and 25,421 questions; the validation set consists of 368 passages and 1,436 questions; the test set consists of 362 passages and 1,436 questions. For the RACE-h subset, the training set consists of 18,728 passages and 62,445 questions; the validation set consists of 1,021 passages and 3,451 questions; the test set consists of 1,045 passages and 3,498 questions. The complete RACE dataset is the RACE-m and RACE-h subsets combined.

The **A**I2 **R**easoning **C**hallenge (ARC) is a multiple-choice QA dataset made up of 7,787 natural science questions from grades 3–9 [16]. The dataset is split into two subsets: the challenge subset and the easy subset, each consisting of 2,590 and 5,197 questions, respectively. For each question, there exist four answer options, with only one being correct.

The challenge subset consists of 1,119 training questions, 299 validation questions, and 1,172 test questions. The easy subset consists of 2,251 training questions, 570 validation questions, and 2,376 test questions. In total there are 3,370 training questions, 869 validation questions, and 3,548 test questions.

The dataset provides a science text corpus of 14 million science sentences related to the task. Information retrieval of sentences from the corpus, or alternative means, is expected for good performance in this task; however, it is not required. We do not utilise any information retrieval in this thesis.

Open **B**ook **Q**uestion **A**nswering (OBQA) is a multiple-choice QA dataset consisting of a corpus of 1,326 elementary level science facts and 5,957 multiple choice questions, each consisting of four answer options, about the corpus [59]; it is modelled after open book exams. The questions are split into 4,957, 500, and 500 training, validation, and test questions, respectively. Alternatively, this dataset can be

converted to a closed book setting where no corpus is utilised; what we do in this thesis.

MCTest is a multiple-choice QA dataset consisting of an array of short fiction stories for children and questions about said stories; all stories have four questions associated with them, each with four answer options [72]. The dataset is split into MC160 and MC500, where the former refers to 160 stories and 640 questions, and the latter refers to 500 stories and 2000 questions. Both sets are randomly split into training, validation, and test sets; MC160's stories are split into 70, 30, and 60 stories, respectively, while MC500's stories are split into 300, 50, and 150 stories, respectively. In this thesis, we utilise both together as a single dataset and report the results on such.

Commonsense**QA** (CQA) is a multiple-choice QA dataset centred around commonsense, constructed to test a model's ability to answer questions that require background knowledge about the world internal to the model [88]. The dataset consists of five multiple-choice answers, the first three being target concepts from ConceptNet [84], the fourth being a distractor from ConceptNet, and the last being an author created distractor. In total the dataset consists of 12,247 questions and is randomly split into an 80/10/10 training, validation, and test split.

Wino**G**rande (WG) is a commonsense multiple-choice QA dataset consisting of pronoun resolution problems, where each question has two possible answers associated with it and is formulated as a fill in the blank task [73]. The dataset consists of different sized training files (we utilise the training file with 40,398 questions, the extra-large version) and each example has a twin, where a trigger word is changed to flip the answer. The validation set has 1,267 questions, while the test set has 1,767 questions; each question in both sets may not have a twin.

Physical **I**nteraction: **Q**uestion **A**nswering (PIQA) is a multiple-choice QA dataset that tests a model's ability to answer physical commonsense questions, even though they do not experience the physical world [7]. The task consists of a question (goal) and two solutions; the model is tasked to predict the correct solution to the goal. In total there are 16,112 training, 1,838 validation, and 3,084 test questions.

Social **IQa** (SIQA) is a multiple-choice QA dataset that tests a model's commonsense reasoning capabilities about social situations, testing their emotional and social intelligence in everyday scenarios [74]. The task consists of a question and three possible answer options. The dataset consists of 33,410 training, 1,954 validation, and 2,224 test questions.

2.6.3 Extractive

The metric used to evaluate extractive (span extraction) datasets is to calculate the F_1-score [97] of the extracted span compared to the correct span. The F_β-score, more generally, is a measure of a model's accuracy on a task utilising both precision and recall. The F_β-score is formally defined as

$$F_\beta = (1+\beta)\frac{precision \times recall}{(\beta^2 + precision) + recall}.$$

In this thesis, we utilise the macro average F_1-score, where the score is computed independently for each class.

Stanford **Qu**estion **A**nswering **D**ataset (SQuAD) is a reading comprehension dataset focused on the extraction of spans of text from a passage according to a question about said passage [69, 70]. Version one and two of the dataset differ in only that

the second includes an additional 53,755 unanswerable questions produced adversarially; we detail statistics for the second version only. The training set consists of 130,319 questions of which 43,498 are negative (unanswerable); 11,873 questions where 5,945 are negative in the validation set; 8,862 questions of which 4,332 are negative in the test set.

Reasoning **O**ver **P**aragraph **E**ffects in **S**ituations (ROPES) is a reading comprehension dataset that requires the application of knowledge gained from a passage to new situations; an answer span is extracted from the passage to answer the question [48]. The dataset consists of 10,924 training, 1,688 validation, and 1,710 test instances.

Quoref is a co-reference resolution dataset focused on the extraction of spans of text from English paragraphs, extracted from Wikipedia [21]. Each example consists of a passage and a question about the passage, where a span is extracted from the passage to answer the question. The dataset consists of 19,299 training, 2,418 validation, and 2,537 test instances.

2.6.4 Abstractive

The metric used to evaluate abstractive datasets is **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation (ROUGE): measures for automatically comparing the quality of a summary by comparison to an optimal summary, which is often human generated [47]. Specifically, the ROUGE-L metric, where L represents the longest common subsequence, is what is utilised as the metric for abstractive datasets in this thesis; we refer the reader to equations 2, 3, and 4 in [47] for an in-depth description. In abstractive datasets, the answer is not necessarily an extracted span, but a generated answer by the model that can include tokens outside the input's vocabulary.

NarrativeQA is a dataset created to encourage deeper comprehension of language by creating questions that are to be answered after reading through a book or movie script in their entirety; alternatively, a version where shorter human-generated summaries replace the books and scripts exists [44]. The dataset consists of 32,747 training, 3,461 validation, and 10,557 test instances. In this thesis, we utilise the version of the dataset with shorter summaries.

Discrete **R**easoning **O**ver the content of **P**aragraphs (DROP) is an adversarially created English reading comprehension dataset; it requires a model to resolve references in a question and the performing of discrete operations like sorting, addition, and subtraction [25]. The dataset consists of 5,565 training passages, each with an average of 13.91 questions; 582 validation passages, each with an average of 16.38 questions; 588 test passages, each with an average of 16.36 questions.

2.6.5 Yes/no

Yes/no datasets require the generation of a *boolean* (i.e., either true/yes or false/no) answer to a question. We utilise the generation of one token to be taken as the model's answer with the metric being the accuracy in terms of the number of correctly predicted samples versus the total number of samples; alternatively known as exact-match accuracy. Other models utilise the generation of a sequence [42] and calculate the F_1-score; we utilise the exact-match accuracy in this thesis.

BoolQ is a dataset of naturally occurring and unexpectedly challenging boolean questions that require non-factoid information and entailment like inference to solve [15]. The dataset consists of 16,000 questions in total: 9,400 for the training, 3,200 for the validation, and 3,200 for the test sets.

2.6.6 Contrast sets

The creation of contrast sets allow for a more in-depth examination of the abilities a dataset is intended to examine, reducing the systematic gaps of the dataset by disallowing good performance by simple decision rules [28]. Contrast sets are created by the manual perturbation of the test instances of datasets around pivot instances in impactful ways that change the correct label. Note, however, that they are different to adversarial examples — which themselves change the input such that the model's prediction changes, but not the correct label — in that they intend to examine if a model's decision boundary is true to the real decision boundary locally around pivot instances by changing the correct label. In this thesis, we utilise contrast sets for BoolQ, DROP, Quoref, and ROPES, each denoted BoolQ-CS, DROP-CS, Quoref-CS, and ROPES-CS, respectively; we refer the reader to [28] for the methodology of the construction of such contrast sets.

Chapter 3

A Simple Entwinement of Neuromodulation and the Transformer

This chapter introduces a simple version of the Neuromodulated Transformer (NeMoT), which entwines neuromodulation with the Transformer architecture. Section 3.1 provides an introduction to this chapter. Section 3.2 delivers preliminary information needed to understand this chapter (for more detail see Chapter 2). Section 3.3 provides an overview of NeMoT and the multi-format training setup. Section 3.4 introduces a simplistic implementation of NeMoT and formally defines it. Section 3.5 details all experiments conducted on NeMoT and is split into five further subsections. Namely, Sections 3.5.1, 3.5.2, 3.5.3, 3.5.4 and 3.5.5, which represent the pre-training of NeMoT, the fine-tuning of NeMoT on individual datasets, the fine-tuning of NeMoT in a GQA setting, the fine-tuning of the generally trained NeMoT on individual datasets, and the fine-tuning of NeMoT with reading strategies on individual datasets, respectively. Section 3.6 provides a discussion of the results and concludes the chapter.

3.1 Introduction

We describe a model to be generally capable of QA if it can at the very least, achieve near-human performance on not just a single QA dataset, but on any possible question that could be posed to it from potentially any dataset or real-world application. Such a single model is expected to perform well on not only the datasets trained on, but other possibly quite distinct datasets where the skills learned and knowledge acquired during training — such as the ability to perform inference and various reasoning capabilities — should generalise. We emphasise that we focus on the situation where the model is fixed after training, not where it continually learns. We coin the term *general question answering* (GQA) for the previously described domain: performance on a wide variety of distinct, possibly strikingly different QA datasets in a closed environment; QA in the real-world domain, where the questions encountered come from a distribution different to that encountered during training; and all that is in-between the closed environment and the real-world. GQA in machines has many benefits to society: aiding researchers by automating redundant tasks, an improved question answering ability of search engines and dialogue systems, medical diagnosis, and possibly a step towards AGI.

The current paradigm in QA consists of the utilisation of a pre-trained language model via the fine-tuning of the pre-trained language model on either individual tasks or a set of tasks [10, 23, 33, 50, 65, 68, 103]. It has been largely successful,

dominating the QA and GQA leaderboards [37, 41, 42, 52, 100, 110, 111]; however, performance in multiple and strikingly distinct datasets has been limited by many factors. For example, the issue of the different formats used in QA datasets and the resulting *echo chamber* [42]; the increasing number of parameters [10, 68] and data [87] needed to achieve better performance in GQA; simple heuristics learned involving entity types, question-context overlap, and learned latent patterns in datasets are robust to corrupt examples (e.g., a shuffled context and incomplete input) but not to authentic variations [77]. Success in improving generalisation has come mainly by increasing the size of QA models [10, 68], training on more data [87], an improved training regime [42, 87], and improvements to the Transformer architecture [18, 67, 111].

We explore neuromodulation in this chapter: a biological mechanism that is involved in the continuous tuning of a neuron's input and output behaviour conditioned on external stimuli in a context-dependent manner [5, 54, 91]. Given the impact that neuromodulation has on organisms, such as the ability to locally modify learning in response to external stimuli, neuromodulation is integrated with ANNs in the hope of achieving the same effects in the continual learning domain. Examples of such include [26, 82, 92] who locally modify the learning rates of individual connections between neurons and [6] who integrates neuromodulation with backpropagation via a gating mechanism.

We hypothesise that neuromodulation will improve performance in the GQA domain because it will allow for the regulation of a population of neurons (activations) [40] conditioned on the context in the form of the provided input to the model itself and additional auxiliary tokens. The input to the model is typically a question with a provided passage, for example. The auxiliary tokens are manually inserted tokens that represent a certain type of question or process. The questions and passages in GQA will likely come from different topics and tasks, which will require different reasoning skills and internal knowledge. The ability to regulate a population of neurons conditioned on the context (i.e., question, passage, and the auxiliary tokens) will allow for more complex rules to be learned, that when coupled with an environment that encourages generalisation (e.g., multi-format training), will improve performance in GQA; it allows for the modification of the output of the model conditioned on the context.

Specifically, we experiment with neuromodulation induced gating in the Transformer architecture via a neuromodulatory mechanism. The neuromodulatory mechanism takes context as input and produces a gating matrix of values between zero and one; through element-wise multiplication, the matrix gates activations at a specific point in another set of Transformer layers. If the value of the gating matrix is one, then the resulting activation is unchanged; as it approaches zero an activation's value is pushed towards zero. Neuromodulation has seen success in the continual learning domain; we aim to see if the success holds in a non-continual learning domain and results in an improved generalisation in QA. We utilise the results in this chapter as a bellwether to see if it should be explored further in not only GQA but more broadly in NLP.

Hence, in this chapter, we extend the Transformer via the entwinement of neuromodulation by introducing the Neuromodulated Transformer (NeMoT). We utilise the text-to-text framework outlined in [42, 68] with NeMoT, as it allows for testing on out-of-domain data of any format; a goal of GQA and this thesis. Additionally, we experiment with integrating two reading strategies with NeMoT: answer option interaction (AOI) [109] and highlighting [86].

3.2 Preliminaries

3.2.1 Transformer

The Transformer is an encoder-decoder architecture that relies entirely on attention, eschewing recurrence entirely [90]. The input sequence to the model is converted into a matrix of word embeddings and has positional information in the form of absolute position embeddings added. The structure of the Transformer consists of an encoder and decoder, each consisting of multiple layers. Each layer consists of multiple blocks, which themselves consist of a residual connection, layer normalization layer, and a module. The role of the encoder is to produce an encoded representation of the input sequence, while that of the decoder is to generate an answer to the task.

In general, a block is traversed in the following order: the module first, residual connection second [32], and layer normalization last [2]. There exist two types of blocks, each differing only in its module. The first block consists of a multi-head attention module, which takes as input a query, key, and value matrix; all are linearly projected and split into h heads. Each head has dot-product attention performed over it, which calculates an attention matrix between the sequence that the query represents and that for the key; if the sequence is the same for both, then it is referred to as self-attention. The second block consists of a point-wise feed-forward network module, which consists of two fully connected layers in a typical neural network; the first layer typically consists of more hidden units than the second.

Each layer in the encoder consists of two blocks, the first containing a multi-head attention module and the second containing a point-wise feed-forward network module. The multi-head attention module performs self-attention, where the attention mechanism consists of a sequence attending to itself. The output of the encoder is referred to as the encoded representation.

Each layer in the decoder consists of three blocks, the first two containing multi-head attention modules and the third containing a point-wise feed-forward network module. The first multi-head attention module performs self-attention like that in the encoder; the second performs not self-attention, but the attending from the decoder's input sequence to that of the encoded representation. After the last layer of the decoder is traversed, its output is passed through a fully connected layer, which generates a prediction over the output vocabulary at each position.

The previously discussed procedure encompasses the Transformer and how it is traversed. For a deeper dive into the Transformer see Section 2.4 in Chapter 2.

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) is a bi-directional masked language model [23]. BERT is practically identical to the encoder in the vanilla Transformer [90] with minor differences: the GELU activation function [34] replaces the ReLU activation function and the input embedding process is modified to include segment embeddings.

The input to BERT is a sequence of text with a classification token $[CLS]$ appended to the beginning of the sequence and a separator token $[SEP]$ that splits the input into segments.

Example. *[CLS] Shaquille O'Neal is the most dominant basketball player ever [SEP] He has broken the backboard many times [SEP]*

The input sequence is converted to a machine readable format by converting all tokens into their associated WordPiece embeddings [98] via a tokenizer that processes the input. As with the vanilla Transformer, positional information is missing, hence, absolute position embeddings are utilised; with the addition of segments via the $[SEP]$ token, each segment has a unique embedding vector associated with it, which is added to every token's embedding vector in the associated segment.

The output of BERT is no different to that in the vanilla Transformer's encoder except for an additional fully connected layer being inserted after the last encoder layer. BERT can perform different tasks: classification via the $[CLS]$ token, tasks that require predictions at each position in the input (e.g., parts of speech tagging), and text generation via the last token in the sequence.

BERT is pre-trained on a masked language modelling and next sentence prediction task. Masked language modelling refers to a task where tokens in a segment of text are masked out, with the goal being to predict the original tokens in the masked positions given the context, i.e., the surrounding tokens. Of the tokens in the sequence, 15% are randomly selected to be masked. Of the masked positions, 80% of the time they are replaced with the $[MASK]$ token, 10% of the time with a random token, and 10% of the time they are unchanged. Next sentence prediction refers to a task where two segments of text, separated by the $[SEP]$ token, are predicted by the $[CLS]$ token position: true if the second segment follows the first and false otherwise.

The pre-trained BERT model can be used as a starting point to fine-tune further on specific tasks, utilising the features learned about language as a starting point and building upon them. At the time of the paper's publication, it achieved SOTA performance on many NLP tasks by fine-tuning on said task [23].

GPT

The **G**enerative **P**re-trained **T**ransformer (GPT) family of architectures [10, 65, 66] involves training a decoder only Transformer on a large chunk of text in an auto-regressive language modelling task. Unlike BERT, GPT models cannot see future tokens in the attention component (i.e., it is uni-directional), simulating the situation the model would encounter in evaluation mode, i.e., the generation of the tokens one by one. Here, we will focus on the GPT architecture present in GPT-2 [66].

Compared to the original Transformer decoder [90], there are some modifications to the architecture: the omission of the second multi-head attention block in all layers (that would otherwise take as input the encoded representation if it existed); in the point-wise feed-forward network, the ReLU activation function is replaced with the GELU activation function [34]; in each block, the layer normalization layer is moved to the beginning.

The input and output of GPT are similar to that of the decoder in the vanilla Transformer. We add that additional special tokens such as the start token <s> and end token </s>, which represent the start and end of a sequence respectively, are incorporated into the vocabulary.

GPT is pre-trained on an auto-regressive language modelling task, where it generates a token one by one conditioned on the previously generated tokens acting as context. The probability of generating a sequence $y = (y_1, y_2, \ldots, y_N)$ is given by

$$\mathbb{P}(y) = \Pi_i^N \mathbb{P}(y_i | y_1, y_2, \ldots, y_{i-1}),$$

where the total probability $\mathbb{P}(y)$ is the product of the probabilities of each token in the sequence being generated given the context of all preceding tokens.

3.2.2 Neuromodulation

The communication of neurons through a synapse is achieved through a process known as neurotransmission. It involves the release of a chemical or electrical signal (i.e., a neurotransmitter). A neurotransmitter is a chemical signal released into the synaptic cleft, it binds to receptors on target cells (neurons), changing their electrical properties and resulting in a large variety of post-synaptic effects [17].

Neuromodulation is closely related to neurotransmission; it consists of neuromodulators, a type of neurotransmitter. Neuromodulation refers to a biological mechanism that is involved in the continuous tuning of a neuron's input and output behaviour conditioned on external stimuli in a context-dependent manner [5, 54, 91]; it entails the ability to change the learning rate of individual connections between neurons [26]; and it plays a key role in the facilitation of learning [22]. Neuromodulators are chemical signals with the ability to locally modify learning rates by either up-regulating or down-regulating them in response to external stimuli [1, 11, 26, 36]. They have spatially distributed and temporally extended effects on recipient neurons, potentially allowing for the regulation of a population of neurons [24, 40, 39, 55, 75].

Given the impact that neuromodulation has on organisms, such as the ability to locally modify learning in response to external stimuli, neuromodulation is integrated with ANNs in the hope of achieving the same effects. One such technique to integrate neuromodulation with ANNs is *Gating*. Gating, in the context of neuromodulation in ANNs, refers to the element-wise multiplication between a gating matrix and target matrix [6], where the gating matrix is produced by a network that is passed external context as input. If an activation in the target matrix is multiplied by one, then it is left unchanged; as the value it is multiplied with approaches zero, the target matrix's activation also approaches zero.

3.2.3 Metacognitive reading strategies

Reading strategies is an expansive term used to characterise the planned and explicit actions that a reader undertakes to help them decipher text to meaning [61, 71]. They are utilized by an individual to improve their reading comprehension and proficiency in reading and are often taught to individuals who lack reading comprehension capabilities (e.g. second language learners and children). "Previewing text before reading", "guessing the meaning of unknown words" and "summarizing text information" are some examples of reading strategies [106].

Metacognition is defined as knowledge about cognition and regulation (i.e., control) of cognition, or alternatively, simply thinking about thinking. It is considered a high-level cognition that can be split into many sub-components, but of two mainly: knowledge and control [4]. The metacognitive component of reading strategies involves an individual's knowledge of reading strategies and their ability to control their usage of reading strategies.

3.2.4 Overview of datasets

In this chapter, we utilise language modelling datasets to pre-train the model, allowing it to acquire linguistic knowledge about language. We utilise multiple-choice, extractive, abstractive, and yes/no type QA datasets to test a model's proficiency in QA and GQA via the fine-tuning of the pre-trained model.

Language modelling datasets are utilised to test a model's linguistic knowledge about language; the metric we use is perplexity [80]. The language modelling

datasets we utilise are C4 [68], LAMBADA [62], WikiText-2 and WikiText-103 [57], and PTB [53, 60].

MQA datasets require a model to select one of the provided answer options to the question; the metric we use is exact-match accuracy for the predicted label. The MQA datasets we utilise are RACE [45], ARC [16], OBQA [59], MCTest [72], CQA [88], WG [73], PIQA [7], and SIQA [74].

Extractive datasets involve a model extracting a span of text from a provided passage; the metric we use is the macro average F_1-score. The extractive datasets we utilise are SQuADv2 [69], ROPES [48], and Quoref [21].

Abstractive datasets are similar to extractive datasets, except for that a model is expected to generate an answer that is not a span of text and consists of tokens outside of the input's vocabulary; the metric we use is ROUGE-L [47]. The abstractive datasets we utilise are NarrativeQA [44] and DROP [25].

Yes/no datasets require the generation of a *boolean* (i.e., either true/yes or false/no) answer to a question; the metric we utilise is exact-match accuracy to a generated boolean token. We utilise only one yes/no dataset, BoolQ [15].

Additionally, we utilise the contrast set [28] versions of some datasets in our experiments. They are denoted by Quoref-CS, ROPES-CS, DROP-CS, and BoolQ-CS for Quoref, ROPES, DROP, and BoolQ, respectively.

3.3 Overview

In this section we detail the text-to-text framework and multi-format training procedure we utilise for GQA and provide an overview of NeMoT.

3.3.1 Text-to-text framework and multi-format training

A roadblock in achieving good performance in GQA is the different formats of QA datasets. By format, we mean the structure of the input (e.g, if the passage comes before the question or not) and how an output is generated (e.g., token generation versus class prediction via a classification token). One way to overcome such is the text-to-text framework utilised by UnifiedQA [42] and introduced in [68]. The framework involves a model taking text as input and generating new text in a common format as output; UnifiedQA extends this framework to QA, where each QA format is converted into a single unified format.

The following example showcases the unified format we utilise, which each QA format is converted into.

Example 3.3.1. <*p*> *...Kobe Bryant is the greatest Laker of all time...*<*q*> *Who is the greatest Laker of all time?* (1) *Magic Johnson* (2) *Kobe Bryant* (3) *Shaquille O'Neil* (4) *Jerry West* (5) *Kareem Abdul-Jabbar* <*sep*> (2) </*s*>

If a passage is included with the question, then it is included first in the sequence with <*p*> preluding it. Next is the question preluded by <*q*>. If the question is multiple-choice then each answer option follows the question, with each answer option being preluded by $(1), (2), \ldots, (m)$ for each of the m answer options, respectively. After all answer options are listed the separator token <*sep*> is employed to distinguish between the question and the generated answer. After the separator token, the answer is either generated during evaluation mode token by token until the end token </s> is reached, or is listed in its entirety during training mode.

Additionally, we utilise the multi-format training procedure utilised in UnifiedQA because of the increased generalisation it showed and its integration with

the text-to-text framework [42]. Let there exist k different formats $F_1, F_2, \ldots, F_k$ in the multi-format training procedure. Each format F_i consists of d_i datasets, each denoted by $D^i_{d_1}, D^i_{d_2}, \ldots, D^i_{d_i}$. Each dataset D^i_j consists of a training set T^i_j and evaluation set E^i_j. A specific subset of the datasets have no training set, i.e., the training set is empty and the evaluation set is non-empty. Denote such datasets with a non-empty training set as *seed datasets* and those with an empty training set as *non-seed datasets*. The purpose of *non-seed datasets* is to test the generalisation capabilities of the model on out-of-domain datasets, while *seed datasets* are utilised for training. We follow [42] and utilise the same *seed datasets*: RACE, SQuADv2, NarrativeQA, BoolQ, ARC, OBQA, and MCTest.

Multi-format training involves the training of all *seed datasets* together in a training pool $\tilde{T}$ where each dataset is evenly represented. The training set pool is constructed via

$$\tilde{T} = \bigcup_{i=1}^{k} \bigcup_{j=1}^{d_i} \{unified_i(q) | q \in T^i_j\},$$

where $q \in T^i_j$ is a question — which may include the passage, answer options, and answer to the question — and $unified_i(q)$ is the conversion of a question q of the ith format into the unified format as previously described. Each question q is included proportional to $1/|T^i_j|$, where $|T^i_j|$ is the size of the training set for the ith format and jth dataset of that format. Therefore, each batch on average will have the same number of training instances from each of the seed datasets.

3.3.2 Overview of NeMoT

The input to NeMoT is a sequence of text such as that in Example 3.3.1, with three auxiliary tokens appended to the beginning of the sequence (we emphasis that no auxiliary tokens are shown in the example and that they are appended before <*p*>). An auxiliary token fulfils the purpose of adding additional context to the input for the neuromodulatory mechanism to be introduced. In NeMoT there are three positions reserved for auxiliary tokens.

The first position of the three auxiliary token positions is reserved for <*cls*>; its role is primarily for the encoder mode but can easily be extended if needed. It is utilized similarly to how it is in BERT, as a classification token [23].

The second position is reserved for the following auxiliary tokens: <*enc*> and <*dec*>, which correspond to the encoder and decoder mode, respectively. The decoder mode differs from the encoder mode only in that it masks future tokens at a given position (i.e., it is uni-directional, not bi-directional). The aim of the neuromodulatory mechanism in relation to the second position's auxiliary token is to gate the forward traversal of another network differently depending on if the model is uni-directional or bi-directional.

The third position involves the following auxiliary tokens: <*lm*>, <*mqa*>, and <*gqa*>; representing language modelling, multiple-choice question answering, and generate question answer — the auto-regressive generation of a sequence of text to answer the question — respectively. The neuromodulatory mechanism will gate the forward traversal of another network differently depending on the auxiliary token in the third position (alternatively known as the *task position*).

NeMoT consists of three components, the *vanilla set*, *neuromodulatory set*, and *output set*. The input sequence as described previously is converted to a machine readable format via a tokenizer that processes the input and converts it into its associated word embedding. The input is represented by a matrix of dimension $\mathbb{R}^{N \times d_{model}}$, where N is the sequence length including the auxiliary tokens, and d_{model} is the dimension of NeMoT and the word embeddings; the resulting matrix has a positional encoding applied to it (e.g., fixed absolute position embedding).

The vanilla set is a typical Transformer that takes as input the word embeddings excluding the auxiliary tokens. Its objective is to process the input without the auxiliary tokens similarly to how a Transformer usually operates. The output is a matrix of the same dimension as the input.

The neuromodulatory set is a typical Transformer that takes as input the output of the vanilla set with the word embeddings of the auxiliary tokens appended to the beginning of the sequence. Its objective is to produce a matrix of values between zero and one to gate the output of the vanilla set before it is passed as input to the output set. By gating the vanilla set's output, it results in the modification of the forward traversal in a context dependent manner.

The output set takes the gated matrix produced by the vanilla set and neuromodulatory set as input. It consists of many parallel Transformer blocks, each corresponding to a specific auxiliary token in the task position; for example, if <*lm*> is the task auxiliary token then the Transformer block that corresponds to language modelling is chosen and traversed. The output of this set is a matrix of the same dimension as the input, which is input to a fully connected layer with a softmax activation function, producing a prediction at each sequence position. For further details on NeMoT, see Section 3.4.

3.4 A simple Neuromodulated Transformer

3.4.1 Introducing NeMoT

NeMoT, as depicted in Figure 3.1, is an extension to the Transformer via the entwinement of neuromodulaion; it is inspired by the gating mechanism in [6]. This version of NeMoT emphasises simplicity, with the objective being to measure the capabilities of a neuromodulatory mechanism in the Transformer. The capabilities are measured by performance in individual QA datasets and in a general QA setting (see Sections 3.5.2 and 3.5.3, respectively).

An emphasis of the design of this architecture is flexibility. For example, it should be able to act in a bi-directional manner like BERT [23] if the user chooses, while easily having the capabilities to be uni-directional like GPT-2 [66]. The neuromodulatory mechanism and the associated auxiliary tokens are designed with this in mind — one can easily add additional auxiliary tokens and modes if desired.

NeMoT consists of three sets of layers: the vanilla set, neuromodulatory set, and the output set. The vanilla set is depicted in Figure 3.1 as the set of Y layers. It receives as input a matrix representing the word embeddings of the input sequence, excluding the auxiliary token positions; denote the length of the sequence excluding the auxiliary tokens as N_{vs}. Positional encodings are added to the input sequence excluding the auxiliary tokens as sketched in the figure.

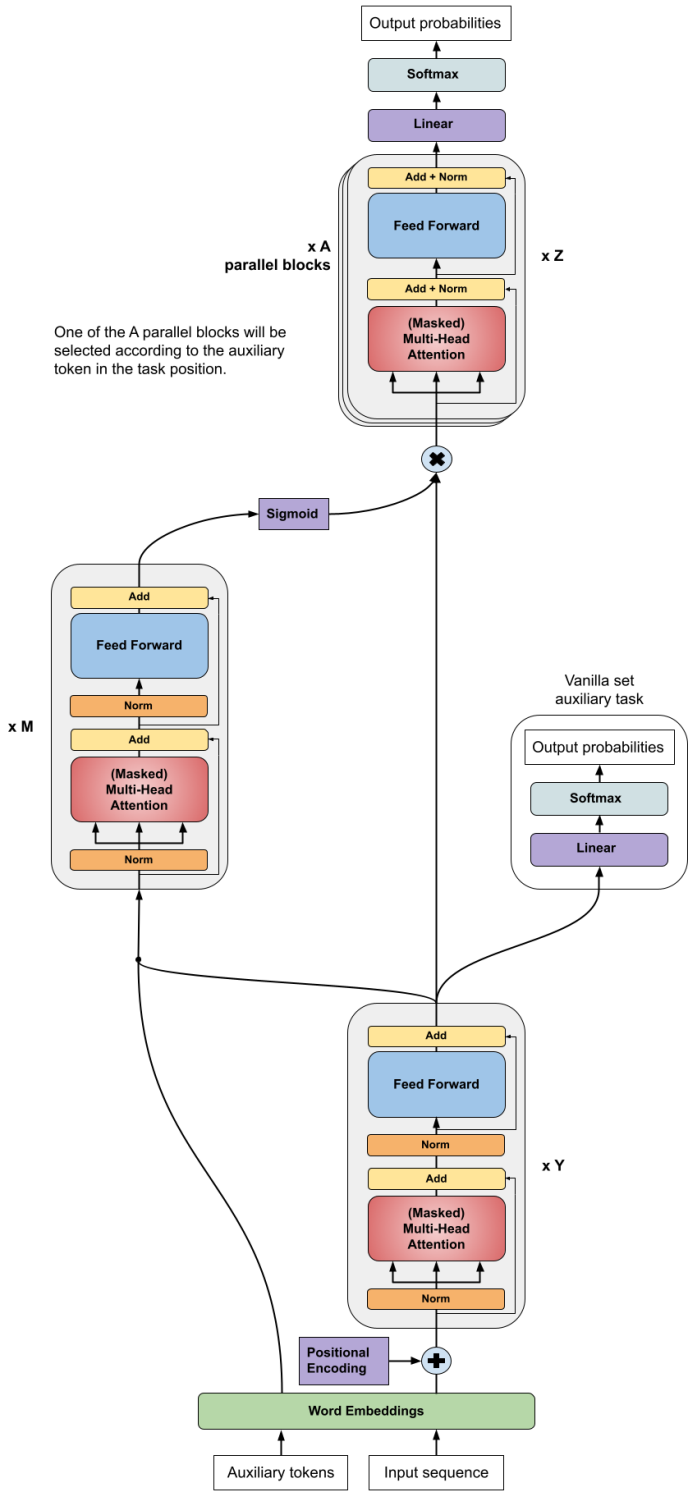


FIGURE 3.1: The Neuromodulated Transformer (NeMoT). The input to the model is an input sequence and auxiliary tokens. Both are converted into their associated word embeddings and absolute position embeddings are applied to only the input sequence, not the auxiliary tokens. The model consists of a vanilla set (the set of Y layers), neuromodulatory set (the set of M layers), and output set (the set of A parallel Transformer blocks, each consisting of Z layers). The vanilla set has its own fully connected layer to generate a prediction for auxiliary tasks, such as generating an auxiliary loss.

Specifically, we utilise fixed absolute position embeddings, defined as follows:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}), \end{aligned} \tag{3.1}$$

where pos is a position in the input sequence, i is the dimension index, and d_{model} is the dimension of the Transformer and the word embeddings. Each dimension i alternates between a sine and cosine function for even and odd dimension indices, respectively, with the wavelengths forming a geometric progression from 2π to $10000 \times 2\pi$.

The neuromodulatory set is the set of M layers. It takes the output matrix of the vanilla set, which is of dimension $N_{vs} \times d_{model}$, and concatenates the auxiliary token word embeddings to the beginning of the matrix, constructing a new matrix of dimension $N_{aux} \times d_{model}$; the length of the sequence including the auxiliary tokens is denoted by N_{aux}. The output matrix of the neuromodulatory set has the sigmoid function applied to it and is used to gate — via values between zero and one, scaling the value of activations — the output matrix of the vanilla set; the auxiliary token positions are excluded to match the shape of the matrices.

The output set is the set consisting of A parallel Transformer blocks, each consisting of Z layers. It takes as input the gated matrix produced by the vanilla set and neuromodulatory set and traverses through one of the A parallel blocks depending on the auxiliary token in the task position. Its output matrix is passed through a fully connected layer with a softmax activation function, inducing a probability distribution at each position in the sequence along the target vocabulary.

Each layer of NeMoT is made up of two blocks, where each block consists of a residual connection [32], layer normalization layer [2], and a module. The module is either a point-wise feed-forward network module or a multi-head attention module.

The multi-head attention module is defined as

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,\ldots,head_h)W^O \\ \text{where } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V), \end{aligned} \tag{3.2}$$

where the input matrices are $Q \in \mathbb{R}^{N_q \times d_q}$, $K \in \mathbb{R}^{N_k \times d_k}$, and $V \in \mathbb{R}^{N_v \times d_v}$, representing the query, key, and value, respectively; the dimension of the query, key, and value is d_q, d_k, and d_v, respectively; the sequence length of the query, key, and value is N_q, N_k, and N_v, respectively. The query, key, and value matrices are linearly projected via dot-product multiplication with $W_i^Q \in \mathbb{R}^{d_{model} \times d_{model}}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_{model}}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_{model}}$, respectively, resulting in new matrices of the same dimensions; d_{model} is the dimension of NeMoT and d_q, d_k, and d_v are always equal to d_{model}. The linearly projected matrices are split into h evenly sized subsets, each of dimension $N_q \times (d_q/h)$, $N_k \times (d_k/h)$, and $N_v \times (d_v/h)$ for the linearly projected query, key, and value, respectively. The ith subsets (represented by QW_i^Q, KW_i^K, and VW_i^V) are input to a dot-product attention module, where the output is a new matrix of dimension $N_q \times (d_v/h)$; denote this by $head_i$. Note that h must be a factor of d_q, d_k, and d_v. To conclude multi-head attention, all heads are concatenated together to form a new matrix of dimension $\mathbb{R}^{N_q \times d_v}$, which is then multiplied with $W^O \in \mathbb{R}^{d_v \times d_{model}}$ via dot-product multiplication, resulting in a new matrix of dimension $\mathbb{R}^{N_q \times d_{model}}$, concluding multi-head attention.

The dot-product attention module's focal point is an attention matrix that induces a probability distribution across rows (i.e., each row sums to one) with each

column representing a token to attend to. It is defined as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{3.3}$$

where the input matrices are $Q \in \mathbb{R}^{N_q \times d_q}$, $K \in \mathbb{R}^{N_k \times d_k}$, and $V \in \mathbb{R}^{N_v \times d_v}$, representing the query, key, and value, respectively. The query matrix is multiplied with the transposed key matrix via dot-product multiplication to produce a new matrix of dimension $N_q \times N_k$, which is then scaled by the square root of the scaling parameter $\sqrt{d_k}$. The softmax activation function is applied across rows to the scaled matrix to produce an attention matrix $A \in \mathbb{R}^{N_q \times N_k}$. The attention matrix induces a probability distribution across rows, i.e., the sequence represented by the query attends to that of the key. Dot-product multiplication is performed between the attention matrix and the value matrix V, producing a new matrix of dimension $N_q \times d_v$, concluding dot-product attention. Note that d_q, d_k, and d_v are always equal; N_k and N_v must always be equal; for self-attention, N_q and N_k must be equal. In practice, some of the tokens in the attention matrix are masked meaning that they cannot be attended to by other tokens (see Figure 3.2 for an overview of masking).

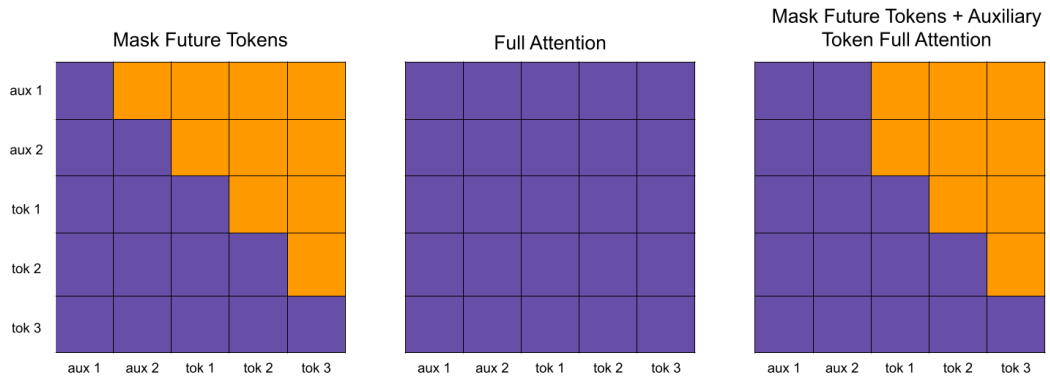


FIGURE 3.2: Types of masking in the attention matrix: purple indicates that attending to that location is allowed, while orange indicates that it is not allowed (i.e., they are masked); "aux" references auxiliary tokens, while "tok" refers to all other tokens. The tokens on the y-axis attend to positions on the x-axis; all attention matrices shown are performing self-attention. In uni-directional models all future tokens are masked; it is displayed in the "Mask Future Tokens" matrix. Bi-directional models utilise full attention, where no positions are masked; it is depicted in the "Full Attention" matrix. A combination of the two, which is utilised by NeMoT, is presented in the "Mask Future Tokens + Auxiliary Token Full Attention" matrix, where the auxiliary token positions have full attention to other auxiliary tokens, but otherwise future tokens are masked.

The point-wise feed-forward network module consists of two fully connected layers, one directly after the other in a typical neural network. It is defined as

$$\begin{aligned} a_1 &= GELU(XW_1) \\ a_2 &= a_1 W_2 \\ FFN_{\text{gelu}}(X) &= a_2, \end{aligned} \tag{3.4}$$

where the input is a matrix $X \in \mathbb{R}^{N \times d_{model}}$, and $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ represent the weight matrices of the first and second fully connected layers respectively; N, d_{model}, and d_{ff} represent the sequence length, dimension of NeMoT, and dimension of the point-wise feed-forward network, respectively. The matrix X is multiplied with W_1 by dot-product multiplication and has the Gaussian Error Linear Unit (GELU) [34] applied directly after, constructing a new matrix $a_1 \in \mathbb{R}^{N \times d_{ff}}$. Dot-product multiplication is performed between a_1 and W_2, producing a new matrix $a_2 \in \mathbb{R}^{N \times d_{model}}$, concluding the point-wise feed-forward network.

The block structure differs in NeMoT's vanilla and neuromodulatory set to that of the vanilla Transformer — unlike the output set — more closely aligning with that of GPT-2 [66]. The point-wise feed-forward network block for the vanilla and neuromodulatory set is

$$\begin{aligned} x_1 &= LayerNorm(X) \\ x_2 &= FFN_{\text{gelu}}(x_1) \\ x_3 &= (X + x_2) \\ pwffn_{\text{block-nmt}}(X) &= x_3, \end{aligned} \tag{3.5}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, and d_{model} is the dimension of NeMoT. The input is first passed through a layer normalization layer, producing a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. The point-wise feed-forward network module (3.4) takes as input x_1 and constructs a new matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$. A residual connection is utilised via the element-wise addition of X and x_2 together, producing a new matrix $x_3 \in \mathbb{R}^{N \times d_{model}}$; it is utilised as the output of the block.

The point-wise feed-forward network block for the output set is

$$\begin{aligned} x_1 &= FFN_{\text{gelu}}(X) \\ x_2 &= LayerNorm(X + x_1) \\ pwffn_{\text{block-oset}}(X) &= x_2, \end{aligned} \tag{3.6}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, and d_{model} is the dimension of NeMoT. The input is first passed through the point-wise feed-forward network module (3.4), generating a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. Layer normalization is applied to the element-wise addition of x_1 and X (i.e., a residual connection), resulting in a new matrix matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$, which is taken as the output of the block.

The multi-head attention block for the vanilla and neuromodulatory set is

$$\begin{aligned} x_1 &= LayerNorm(X) \\ x_2 &= MultiHead(x_1, x_1, x_1) \\ x_3 &= X + x_2 \\ mha_{\text{block-nmt}}(X) &= x_3, \end{aligned} \tag{3.7}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, and d_{model} is the dimension of NeMoT. NeMoT's architecture only utilises self-attention, meaning that only a single matrix X is needed as input. The input matrix is passed through a layer normalisation layer, constructing a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. The multi-head attention module (3.2) performs self-attention by utilising x_1 as the query, key, and value, cultivating a new matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$. A residual connection is performed by the element-wise addition of X and x_2, producing a new matrix

$x_3 \in \mathbb{R}^{N \times d_{model}}$; it is utilised as the output of the block.

The multi-head attention block for the output set is

$$\begin{aligned} x_1 &= MultiHead(X, X, X) \\ x_2 &= LayerNorm(X + x_1) \\ mha_{\text{block-oset}}(X) &= x_2, \end{aligned} \tag{3.8}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, and d_{model} is the dimension of NeMoT. The input is first passed through the multi-head attention module (3.2) as the query, key, and value, generating a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. Layer normalization is applied to the element-wise addition of X and x_1 (i.e., a residual connection), resulting in a new matrix matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$, which is taken as the output of the block.

There exists two types of layers in NeMoT: the first consists of blocks that have the layer normalisation layer at the beginning of the block (see equations in (3.7) and (3.5)) and the second consists of blocks where the layer normalization layer is after the residual connection, at the end of the block (see equations in (3.8) and (3.6)).

A layer in the vanilla and neuromodulatory set consists of two blocks, where all blocks are the version where layer normalization has been moved to the beginning. It is defined as

$$\begin{aligned} x_1 &= mha_{\text{block-nmt}}(X) \\ x_2 &= pwffn_{\text{block-nmt}}(x_1) \\ NeMoT^i_{\text{layer-nmt}}(X) &= x_2, \end{aligned} \tag{3.9}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, d_{model} is the dimension of NeMoT, and i represents the ith layer. The input matrix X is used as input to the multi-head attention block (3.7), returning a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. The point-wise feed-forward block (3.5) takes as input x_1 and produces a new matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$; it is taken as the output of the layer.

A layer in the output set consists of two blocks, where all blocks are the version where layer normalization is directly after the residual connection. It is defined as

$$\begin{aligned} x_1 &= mha_{\text{block-oset}}(X) \\ x_2 &= pwffn_{\text{block-oset}}(x_1) \\ NeMoT^i_{\text{layer-oset}}(X) &= x_2, \end{aligned} \tag{3.10}$$

where the input matrix is $X \in \mathbb{R}^{N \times d_{model}}$, N is the sequence length of the input, d_{model} is the dimension of NeMoT, and i represents the ith layer. The input matrix X is used as input to the multi-head attention block (3.8), returning a new matrix $x_1 \in \mathbb{R}^{N \times d_{model}}$. The point-wise feed-forward block (3.6) takes as input x_1 and produces a new matrix $x_2 \in \mathbb{R}^{N \times d_{model}}$; it is taken as the output of the layer.

The vanilla set, and all Y layers are defined as

$$\begin{aligned} O^1_{vset} &= NeMoT^1_{\text{layer-nmt}}(X_0) \\ O^i_{vset} &= NeMoT^i_{\text{layer-nmt}}(O^{i-1}_{vset}) \; \forall \{1 < i \leq Y\} \\ NeMoT_{\text{vanilla-set}}(X_0) &= O^Y_{vset}, \end{aligned} \tag{3.11}$$

where the input matrix is $X_0 \in \mathbb{R}^{N_{vs} \times d_{model}}$, N_{vs} is the length of the input sequence excluding the auxiliary tokens, and d_{model} is the dimension of NeMoT. The

first vanilla set layer $NeMoT^1_{\text{layer-nmt}}$ takes as input X_0 and produces a new matrix $O^1_{vset} \in \mathbb{R}^{N_{vs} \times d_{model}}$. For all subsequent layers, the input to the ith layer is the output of the previous layer O^{i-1}_{vset}; a new matrix $O^i_{vset} \in \mathbb{R}^{N_{vs} \times d_{model}}$ is generated. When the Yth layer's output matrix O^Y_{vset} is generated it is taken as the output of the vanilla set.

The neuromodulatory set and all M layers are defined as

$$\begin{aligned} O^1_{nmset} &= NeMoT^1_{\text{layer-nmt}}([X_{aux}; O^Y_{vset}]) \\ O^i_{nmset} &= NeMoT^i_{\text{layer-nmt}}(O^{i-1}_{nmset}) \, \forall \, \{1 < i \leq M\} \\ NeMoT_{\text{nm-set}}(X_{aux}, O^Y_{vset}) &= sigmoid(O^M_{nmset}), \end{aligned} \tag{3.12}$$

where the input matrices are $O^Y_{vset} \in \mathbb{R}^{N_{vs} \times d_{model}}$ and $X_{aux} \in \mathbb{R}^{\ell_{aux} \times d_{model}}$, representing the output of the vanilla set and the word embeddings of all $\ell_{aux} \in \mathbb{R}$ auxiliary tokens, respectively; d_{model} is the dimension of NeMoT, N_{vs} is the length of the input sequence excluding the auxiliary tokens, and N_{aux} is the length of the input sequence including the auxiliary tokens. The first layer of the neuromodularity set $NeMoT^1_{\text{layer-nmt}}$ takes as input the concatenation of both input matrices ($[X_{aux}; O^Y_{vset}]$) along the sequence dimension and produces a new matrix $O^1_{nmset} \in \mathbb{R}^{N_{aux} \times d_{model}}$, where N_{aux} is the length of the sequence including the auxiliary tokens. For all subsequent layers, the input to the ith layer is the output of the previous layer O^{i-1}_{nmset}; a new matrix $O^i_{nmset} \in \mathbb{R}^{N_{aux} \times d_{model}}$ is generated. When the Mth layer's output matrix O^M_{nmset} is generated it is taken as the output of the neuromodulatory set after having the sigmoid function applied to it, converting all values in the matrix to a value between zero and one.

The output set consists of A parallel Transformer blocks, of which one is chosen and traversed depending on the task auxiliary token. Here, we will detail the output set for one of the A parallel Transformer blocks, but keep in mind that any of the A parallel blocks can be chosen. The output set and all Z layers are defined as

$$\begin{aligned} O^1_{oset} &= NeMoT^1_{\text{layer-oset}}(X_{gate}) \\ O^i_{oset} &= NeMoT^i_{\text{layer-oset}}(O^{i-1}_{oset}) \, \forall \, \{1 < i \leq Z\} \\ NeMoT_{\text{output-set}}(X_{gate}) &= O^Z_{oset}, \end{aligned} \tag{3.13}$$

where the input is the gated matrix $X_{gate} \in \mathbb{R}^{N_{vs} \times d_{model}}$, N_{vs} is the sequence length excluding the auxiliary tokens, and d_{model} is the dimension of NeMoT. The first layer $NeMoT^1_{\text{layer-oset}}$ takes as input X_{gate} and manufactures a new matrix $O^1_{oset} \in \mathbb{R}^{N_{vs} \times d_{model}}$. For all subsequent layers, the input to the ith layer is the output of the previous layer O^{i-1}_{oset}; a new matrix $O^i_{oset} \in \mathbb{R}^{N_{vs} \times d_{model}}$ is generated. When the Zth layer's output matrix O^Z_{oset} is generated it is taken as the output of the output set.

NeMoT consists of the three previously defined sets and an additional fully connected layer with a softmax activation function applied across the target vocabulary (i.e., for each position in a sequence a probability distribution is induced over the

target vocabulary). It is defined as

$$\begin{aligned} O^{Y}_{vset} &= NeMoT_{\text{vanilla-set}}(X_{inp}) \\ O^{M}_{nmset} &= NeMoT_{\text{nm-set}}(X_{aux}, O^{Y}_{vset}) \\ X_{gate} &= O^{M}_{nmset}[\ell_{aux}+1:,:] \times O^{Y}_{vset} \\ O^{Z}_{oset} &= NeMoT_{\text{output-set}}(X_{gate}) \\ O_{logits} &= O^{Z}_{oset} \cdot W_{output} \\ O_{prob} &= Softmax(O_{logits}) \\ NeMoT(X_{aux}, X_{inp}) &= O_{prob}, \end{aligned} \tag{3.14}$$

where the input matrices are $X_{aux} \in \mathbb{R}^{\ell_{aux} \times d_{model}}$ and $X_{inp} \in \mathbb{R}^{N_{vs} \times d_{model}}$, representing the auxiliary tokens and input sequence, respectively; ℓ_{aux} is the number of auxiliary tokens, N_{vs} is the length of the input sequence excluding the auxiliary tokens, and N_{aux} is the length of the input sequnce including the auxiliary tokens. The vanilla set (3.11) takes as input X_{inp} and generates a new matrix $O^{Y}_{vset} \in \mathbb{R}^{N_{vs} \times d_{model}}$. Both X_{aux} and O^{Y}_{vset} are utilised as input to the neuromodulatory set (3.12); a new matrix $O^{M}_{nmset} \in \mathbb{R}^{N_{aux} \times d_{model}}$ is constructed by the set.

Gating is performed between the output matrices of the neuromodulatory set and the vanilla set via an element-wise multiplication; the former's matrix is indexed ($[\ell_{aux}+1:,:]$) to exclude the auxiliary token positions to match the sequence length dimension of the vanilla set's output matrix. Note that for $[\ell_{aux}+1:,:]$, the indexing starts at 1, not 0 and that the lower argument of the range is inclusive, while the upper is exclusive. A new matrix $X_{gate} \in \mathbb{R}^{N_{vs} \times d_{model}}$ is generated by gating.

The output set takes as input the gated matrix X_{gate}; a new matrix $O^{Z}_{oset} \in \mathbb{R}^{N_{vs} \times d_{model}}$ is produced by one of the set's A parallel blocks, conditioned on the auxiliary token in the task position. The output matrix of the output set is passed as input to a fully connected layer with the softmax activation function (i.e., dot-product multiplication between $O^{Z}_{oset} \in \mathbb{R}^{N_{vs} \times d_{model}}$ and $W_{output} \in \mathbb{R}^{d_{model} \times |V_{tar}|}$ is performed and the softmax function is applied to the resulting matrix) assembling a new matrix $O_{prob} \in \mathbb{R}^{N_{vs} \times |V_{tar}|}$, where V_{tar} is the target vocabulary and $|V_{tar}|$ represents the size of the target vocabulary. A probability distribution is induced across the target vocabulary at each position in the sequence, with the maximum value's associated token being chosen as the prediction.

3.4.2 Design choices

When designing the neuromodulatory mechanism, we made the design choice that the vanilla set's output matrix is to be input to the neuromodulatory set. Therefore, now when the neuromodulatory mechanism generates the gating matrix to gate the vanilla set's output matrix, it is not doing so independent of it. The consequence of such is that during training the vanilla set's parameters are being optimised for two paths: the generation of the gating matrix and the output of NeMoT through the output set.

We recognise that the gradient flowing back through the vanilla set from the neuromodulatory set may harm what information the vanilla set's output matrix encodes in terms of its contribution to the output set and thus the output of NeMoT. Therefore, in an attempt to mitigate the concerns, in our experiments we couple an auxiliary loss with the vanilla set's output matrix after it is passed through a fully

connected layer with a softmax activation function to generate a prediction for the current task (see the "Vanilla set auxiliary task" pathway in Figure 3.1).

The auxiliary tokens are only utilised as input to the neuromodulatory set because their purpose is to act as context and modify the gating matrix accordingly. Additionally, they fulfil a second objective of choosing which Transformer block in the output set to traverse via the task auxiliary token.

Given that we are utilising multi-format training, we introduce modularity into the output set via the traversal of one of the parallel Transformer blocks. The purpose of the introduced modularity is to reduce interference between different types of datasets (e.g., multiple-choice versus extractive) in the last layers before a prediction is generated.

3.4.3 Reading strategies in NeMoT

We experiment with two reading strategies in NeMoT: the first is answer option interaction (AOI), first introduced in [109] whose implementation we modify; the second is highlighting, which we extend [86]'s implementation. An analogy as to why AOI might be useful in multiple-choice question answering is that if you know that a subset of answers is not correct, you are either left with the correct answer or fewer answer options to take a random guess; you can increase your accuracy in a set of MQA questions by just eliminating impossible answers, even if you do not know the correct answer. For highlighting in machines, it aims to simulate the situations when humans highlight important parts of a text, drawing attention to them.

A problem with the initial implementation of AOI in [109] is that it needs to be computed many times between each pair of answer options. We simplify the method so it only needs to be computed once. We formulate AOI in NeMoT as

$$\begin{aligned} H_1 &= W_1 X_{ans} \\ H_2 &= W_2 X_{ans} \\ G &= Softmax(H_1 W_3 H_2^T) \\ H_{int} &= ReLU(G H_2) \\ g &= sigmoid([H_{int}; H_1] W_4) \\ X_{aoint} &= (g \times H_1) + ((1-g) \times H_{int}) \\ AOI(X_{ans}) &= X_{aoint}, \end{aligned} \tag{3.15}$$

where $W_1, W_2, W_3 \in \mathbb{R}^{d_{model} \times d_{model}}$, $W_4 \in \mathbb{R}^{2d_{model} \times d_{model}}$, $X_{ans} \in \mathbb{R}^{N_{ans} \times d_{model}}$ is the input matrix containing only the answer option positions, N_{ans} is the input sequence length (of the answer options), and d_{model} is the dimension of NeMoT. The input matrix X_{ans} is linearly projected via W_1 and W_2, producing two new matrices H_1 and H_2, each of dimension $N_{ans} \times d_{model}$. A bilinear interaction matrix $G \in \mathbb{R}^{N_{ans} \times N_{ans}}$ is constructed by calculating the dot-product between H_1 and W_3, whose resulting matrix then has the dot-product calculated between it and the transpose of H_2; the Softmax activation is applied along the row dimension (i.e., each row sums to one). The interaction representation $H_{int} \in \mathbb{R}^{N_{ans} \times d_{model}}$ is produced by the dot-product multiplication between G and H_2 after being passed through a ReLU activation function. A reset gate $g \in \mathbb{R}^{N_{ans} \times d_{model}}$ is manufactured via applying the sigmoid function to the dot-product between the concatenation of H_{int} and H_1 (resulting in a new matrix of dimension $N_{ans} \times 2d_{model}$), and W_4; the reset gate balances the influence of the interaction matrix and the first linearly projected matrix. Information between both

H_1 and H_{int} is merged via the utilisation of the gating matrix, producing a new matrix $X_{aoint} \in \mathbb{R}^{N_{ans} \times d_{model}}$; it is taken as the output of the AOI module. Lastly, we note that all bias terms have been omitted for W_1, W_2, W_3, and W_4; dropout is applied to X_{aoint}.

We extend the highlighting reading strategy in [86]: it consists of vectors ℓ^+ and ℓ^- (of dimension d_{model}), which are added to a document embedding. If a position's token corresponds to a POS tag that associates with a noun, verb, adjective, adverb, numeral, or foreign word then ℓ^+ is added at that position, otherwise, ℓ^- is added. Our extension involves extending the number of vectors from two to eight, where nouns, verbs, adjectives, adverbs, numerals, foreign words, and the other category each have their own vector; additionally, the eighth vector is reserved for special tokens. They apply the vectors to the document (i.e., passage) component of the input, while we apply it to the entire input, including the question and answer options.

Both reading strategies can be applied at any point in the network, with right after the embedding layer being the most logical. However, due to ease of implementation in our code we apply them to the output of the vanilla set; we note that performance is likely going to be better if we apply them right after the embeddings so the vanilla set can utilise the effects of the strategies. We are still expecting an improvement in performance with the strategies when applied to the output of the vanilla set. We utilise two variants of reading strategies in NeMoT: the first is AOI, which applies AOI to the vanilla set's output, producing a new matrix of the same dimension to take its place; the second is highlighting, where the highlighting vectors are added to the vanilla set's output.

3.5 Experiments

The objective of the experiments section is to determine if neuromodulation can be beneficial not only in fine-tuning on individual datasets, but in a GQA setting, and whether or not neuromodulation should be explored further; additionally, we experiment with two reading strategies. To test such we pre-train a uni-directional NeMoT in Section 3.5.1 in an auto-regressive language modelling task. We determine its capabilities in QA on individual datasets by comparing it to a baseline of relatively similar size and structure, but without neuromodulation, in Section 3.5.2. We utilise the same baseline model as a comparison to NeMoT in a GQA scenario in Section 3.5.3, i.e., we generally train NeMoT and the baseline model. In Section 3.5.4 we test if the generally trained NeMoT is a better starting point for fine-tuning on individual datasets than the pre-trained version. Lastly, we integrate reading strategies with the generally trained NeMoT and test their contribution to performance on MQA datasets in Section 3.5.5.

3.5.1 Pre-training

Pre-training is a fundamental component of the current paradigm, with the knowledge acquired during pre-training being transferred to other tasks via fine-tuning. Specifically, in this section, we detail the hyper-parameters and pre-training procedure of NeMoT. We evaluate the performance of the pre-trained NeMoT versus a baseline in language modelling.

We code NeMoT in the Python programming language, utilising the Tensorflow and Keras libraries. It follows the description in Section 3.4 with two small additions: a layer normalization layer directly after the output set and dropout layers in each

block before the residual connection. All layer normalization layers are initialised with epsilon set to 1e-5, while during training the dropout layers have a dropout rate of 0.1. All fully connected (dense) layers have an L2 kernel regularizer with the regularizer factor set to 0.01. All parameters in NeMoT are initialised via the Xavier uniform initializer [29].

The model itself consists of 12 vanilla set layers, 12 neuromodulatory set layers, and 6 parallel Transformer blocks in the output set — of which only one is utilised during pre-training, the one associated with the language modelling auxiliary token <*lm*> — each consisting of 3 layers. The dimension of NeMoT and the word embeddings is 768, the dimension of the point-wise feed-forward network is 3072, and each multi-head attention module consists of 12 heads.

Fixed absolute position embeddings (3.1) are utilised and added to an input sequence's word embedding matrix; each input sequence is padded to a length of 768, or of length 771 including the three auxiliary tokens. The GPT-2 tokenizer, which is a Byte Pair Encoding (BPE) [78] from the Hugging Face library, is utilised; it consists of a vocabulary of 50257 token[1]. The auxiliary tokens and any additional special tokens (such as "(1)" and "<*sep*>", for example) are added to the vocabulary, resulting in a new vocabulary size of 50313. As we are traversing English only and have no intent on exploring any other language in this thesis, the input and target vocabularies are the same and both utilise the same tokenizer.

The dataset utilised for pre-training is the **C**olossal **C**lean **C**rawled **C**orpus (C4) [68]. It consists of a vast amount of clean English text scraped from the web (750 GB), more than we can realistically train on given our computational resources[2]. It is a high-quality dataset that, in the Hugging Face extracted version, is conveniently split into many sub-files, each of approximately 820 MB; we randomly sample without replacement from the training files during training until the desired number of iterations (1 million) has been achieved.

The three auxiliary tokens utilised during pre-training are fixed. In order, they are <*cls*>, <*dec*>, and <*lm*>, representing the classification token, decoder mode token (future tokens are masked), and auto-regressive language modelling token, respectively.

Given the large amount of computational resources needed to train moderately large language models, our limited computational resources to do so and our desire for such, we choose the pre-training procedure in accordance. One thing to consider is the pre-training task: masked language modelling [23, 50] or auto-regressive language modelling [10, 65, 66]. Masked language modelling entails 15% of a sequence's tokens being masked and thus predictable; auto-regressive language modelling entails all positions being predictable and the restriction of a token from seeing future tokens. We choose auto-regressive language modelling with a uni-directional NeMoT because we hypothesis that 100% of positions being predictable versus the 15% of masked language modelling with a bi-directional NeMoT will train faster and on less data (subject to debate on whether or not auto-regressive language modelling might be harder to learn because a position cannot see future positions unlike the masked positions in masked language modelling; potentially resulting in a longer training time for auto-regressive language modelling). In utilising a uni-directional model we hurt performance on downstream QA tasks as future tokens will be masked unnecessary; however, there exist other similar models such as GPT to act as a comparison, more specifically GPT-2 Medium [66] in our experiments.

[1] https://huggingface.co/docs/transformers/model_doc/gpt2

[2] We extract a compressed version through Hugging Face: https://huggingface.co/datasets/c4

Additionally, we decide to utilise an already pre-trained GPT-2 Small model [66] as the vanilla set as it was observed to speed up pre-training. GPT-2 is structurally similar to our originally intended vanilla set and in essence, it is what our vanilla set would end up like had it been pre-trained similarly to that of GPT-2 small. GPT-2 Small is pre-trained with a sequence length of 1024, therefore, when acting as the vanilla set its input sequence length is shrunk to 768 to match that of NeMoT.

We decide to pre-train for 1 million iterations on randomly sampled training sub-files of C4. A batch size of 32 is utilised, split evenly across four Quadro RTX 8000 GPUs; training commenced for approximately three weeks. Categorical cross entropy, cosine decay [51] with an initial learning rate of 1e-4 and 1,000,000 decay steps, and the ADAM optimizer [43] with default parameters is employed.

In total, approximately 58 GB of data was utilised during pre-training, notably more than that used to pre-train the GPT-2 models[3]. Figure 2 in [65] shows that the fewer layers that are transferred from pre-training to downstream QA tasks the worse the performance is, suggesting that the quality of the pre-trained model may have an impact on performance on downstream QA tasks (i.e., performance after fine-tuning). Hence, more pre-training time would likely be beneficial and is something we should keep in mind when discussing the experiments.

Models	Datasets			
	LAMBADA (PPL)	Wikitext-2 (PPL)	Wikitext-103 (PPL)	PTB (PPL)
GPT-2 Small	35.13	29.41	37.50	65.85
GPT-2 Medium	15.60	22.76	26.37	47.33
NeMoT$_{200}$♣	71.33	36.91	33.32	36.24
NeMoT$_{183}$♣	71.25	36.96	33.36	36.25
NeMoT$_{173}$♣	71.77	36.94	33.40	36.35
NeMoT$_{155}$♣	71.85	37.26	33.66	36.53
NeMoT$_{130}$♣	73.48	38.01	34.56	38.05
NeMoT$_{111}$♣	75.86	38.72	35.37	37.72
NeMoT$_{200}$♠	71.33	30.81	27.77	31.89

TABLE 3.1: Zero-shot language modelling on four language modelling datasets' test sets. For all models, no further training is performed outside of pre-training. The reported metric is perplexity (PPL) for all datasets. GPT-2 Small and GPT-2 Medium's results are taken from [66]. NeMoT models with a ♠ and ♣ represent sliding windows of size 32 and 768, respectively; the GPT-2 models utilise a sliding window of 1024. NeMoT$_x$ refers to the xth checkpoint, where each checkpoint is saved after 5000 iterations.

Here, we compare the performance of the pre-trained NeMoT in a zero-shot language modelling setting (i.e., no further training) to GPT-2 Small and GPT-2 Medium [66]; the results are portrayed in Table 3.1. We report results on four datasets' test sets: LAMBADA [62], WikiText-2 [57], WikiText-103 [57], and PTB [53, 60]. The GPT-2 models have had tokenization artefacts such as shuffled sentences, contractions, the unknown token, and disconnected punctuation removed when reporting their

[3]Note that this does not include the 40 GB used to pre-train the acting vanilla set, GPT-2 Small. Additionally, it is important to note that the training quality of GPT-2 Small is limited by the relatively small number of parameters in comparison to say GPT-2 Medium, thus, the pre-training data, while identical, is not equivalent between the two models.

results, claiming an increase in perplexity of 2.5 to 5; in their training data, they remove Wikipedia articles, introducing some bias into the results. We remove some small tokenization artefacts such as heading patterns, the new line character, and the unknown token in WikiText; the new line character only in PTB; the removal of the new line character, and ‘ and ’ in LAMBADA. We note briefly that GPT-2 Small and GPT-2 Medium have a larger sequence length than NeMoT (1024 versus 768, respectively), giving it a larger context; they utilise a sliding window of size 1024.

Transformer based models split the input into segments when their sequence length is greater than the sequence length of the model. A disadvantage of such is that in the second segment onwards the earlier tokens have their context cut, degrading performance. A sliding window is a technique that only calculates the perplexity for the last w tokens in a segment. The window is shifted to the right by w in each segment, allowing tokens in the window to have more context available in comparison to if they happened to be at the beginning of a segment. The smaller the sliding window w, the more context that is available to generate a score, resulting in better performance. Note, however, that more computational resources will be needed to generate a score as more segments will need to be processed; a trade-off is required between performance and processing time.

On LAMBADA, NeMoT reports a perplexity of 71.33 on both the 32 and 768 sliding window sizes; much worse than GPT-2 Small and GPT-2 Medium, reporting a perplexity of 35.13 and 15.60, respectively. The length of the examples in LAMBADA is relatively small, approximately 75 words, therefore, the difference in sequence length plays no role in the difference in perplexity.

On WikiText-2, NeMoT achieves a perplexity of 36.91 and 30.81 for sliding windows of size 768 and 32, respectively. GPT-2 Small and GPT-2 Medium obtain a perplexity of 29.41 and 22.76, respectively, each having a sliding window of size 1024. This dataset’s examples are longer than that in LAMBADA and are converted into many segments. Because NeMoT with a sliding window of size 32 performs worse than GPT-2 Small with a sliding window of 1024, it suggests that NeMoT performs relatively poorly.

On WikiText-103, NeMoT achieves a perplexity of 33.32 and 27.77 for sliding windows of size 768 and 32, respectively. GPT-2 Small and GPT-2 Medium obtain a perplexity of 37.50 and 26.37, respectively, each having a sliding window of size 1024. NeMoT performs better than GPT-2 Small for both sliding window sizes, with a better perplexity with a sliding window of size 768 versus 1024 for that of GPT-2 Small (i.e., with less context than GPT-2 Small); NeMoT still performs worse than GPT-2 Medium, even with a sliding window of size 32.

On PTB, NeMoT obtains a perplexity of 36.24 and 31.89 for sliding windows of size 768 and 32, respectively. GPT-2 Small and GPT-2 Medium achieve a perplexity of 65.85 and 47.33, respectively, each having a sliding window of size 1024. Contrary to the other datasets, NeMoT performs much better than GPT-2 Small and GPT-2 Medium. With a sliding window size of size 768, it has a smaller context than that of GPT-2 Small and GPT-2 Medium but performs much better with a perplexity 11.09 lower than that of GPT-2 Medium.

Overall, GPT-2 Medium is the best language model in a zero-shot setting, only being outperformed in one instance by NeMoT on PTB. GPT-2 Small performs better than NeMoT on LAMBADA and WikiText-2; NeMoT performs better on WikiText-103 and PTB. Given that WikiText-2 and WikiText-3’s test sets are nearly identical, with the only difference being WikiText-2 has a smaller vocabulary and thus more unknown tokens, which we remove, it suggests that NeMoT is more sensitive to missing context. This may be because GPT-2 Small’s pre-training data has had

Wikipedia articles removed while NeMoT's pre-training data has not; NeMoT may be more perplexed about missing context because of it.

We report results for various checkpoints of NeMoT to show that it is still improving, even at checkpoint 200 (i.e., at 1 million iterations). NeMoT$_x$ refers to the x^{th} checkpoint's results; we only report results for older checkpoints with a sliding window of size 768. In most instances later checkpoints — with the exceptions of checkpoint 130 on PTB, checkpoint 183 on WikiText-2, and checkpoint 200 on LAMBADA — perform better than previous checkpoints, suggesting that NeMoT can still be improved with more training data, supporting the notion that NeMoT has not yet converged and that more training iterations will be beneficial; however, note that the same may be true for the GPT-2 models.

3.5.2 Fine-tuning on individual datasets

While the main objective of NeMoT is to improve performance in GQA, for that to be achieved a model needs to perform well on individual QA datasets. Hence, in this section we test the model's ability to perform QA on individual datasets, i.e., only utilising a dataset's provided training data for fine-tuning. We measure the performance of NeMoT against a baseline model of similar size and structure (GPT-2 Medium [66]), evaluating the contribution to performance by the neuromodulatory mechanism.

We utilise the following four datasets in this section to compare NeMoT to a baseline model: RACE [45], SQuADv2 [69], NarrativeQA [44], and BoolQ [15]. Each represents one of the four QA categories: multiple-choice, extraction, abstraction, and yes/no, respectively. We formulate the datasets as described in Section 3.3: RACE (multiple-choice) is trained to generate only one token, the correct label; all the others (extraction, abstraction, and yes/no) are trained to generate a sequence of tokens until the end token is produced. Additionally, four other datasets are utilised for a deeper dive into MQA for NeMoT only: CommonsenseQA (CQA) [88], Physical Interaction QA (PIQA) [7], MCTest [72], and WG [73].

Model	Parameters	Layers	Dimension	FFN dimension	Heads	Pre-training data
NeMoT	261M	27	768	3072	12	58 GB*
GPT-2 Medium	345M	24	1024	4096	16	40 GB

TABLE 3.2: A comparison of NeMoT and GPT-2 Medium: parameters, layers, dimension, ffn dimension, heads, and pre-training data represent the number of parameters, the number of layers, the dimension of the model, the dimension of the point-wise feed-forward network, the number of heads in multi-head attention, and the amount of data trained on during pre-training in gigabytes (GB), respectively. "M" represents millions. *Not including the 40 GB of data that GPT-2 Small, the acting vanilla set was pre-trained on.

There exists some differences between NeMoT and the baseline model, GPT-2 Medium; Table 3.2 displays the differences. NeMoT consists of approximately 84 million fewer parameters and was pre-trained on approximately 18 GB more data than GPT-2 Medium, excluding the data used to pre-train the vanilla set, GPT-2 Small (40 GB). Note that the data used to train GPT-2 Small is the same as that for GPT-2 Medium, but the information learned by the two models is not equivalent with GPT-2 Medium achieving much better performance in the zero-shot setting on

other language modelling tasks [66]. Additionally, we utilise the GPT-2 Medium pre-trained model from Hugging Face, which excludes the final fully connected layer; we initialise randomly a fully connected layer of the same dimensions to replace it. Both models are provided with inputs of 768 tokens.

While GPT-2 Medium is not a perfect model to use as a baseline because it consists of a higher dimension and 84 million more parameters, it still serves as a good comparison to NeMoT and the contribution of the neuromodulatory mechanism: if NeMoT's performance is equal to or better than the baseline's, then the neuromodulatory mechanism is considered a net plus; if NeMoT's performance is lagging behind the baseline's a moderate amount, then we can conclude that the difference may come down to the additional parameters of the baseline; if NeMoT's performance is significantly worse, then we can chalk it up to NeMoT being an utter failure. Again, we need to keep in mind that the vanilla set has been trained on an additional 40 GB of data, the final fully connected layer of GPT-2 Medium is randomly initialised, and that GPT-2 Medium achieves better perplexity scores than NeMoT in the zero-shot language modelling setting (see Section 3.5.1) when discussing the results and coming to conclusions.

Model	Dataset	Batch size	Starting loss	Warm-up loss	Warm-up steps	Decay loss	Decay steps	Lm aux loss	Vset aux loss
NeMoT	RACE	32	1e-5	N/A	N/A	N/A	N/A	True	True
NeMoT	SQuADv2	32	5e-5	1e-4	2000	1e-5	120,000	True	True
NeMoT	BoolQ	16	5e-5	1e-4	250	1e-5	18,000	True	True
NeMoT	NarrativeQA	32	5e-5	1e-4	2000	1e-5	120,000	True	True
GPT-2 Medium	RACE	32	1e-5	N/A	N/A	N/A	N/A	True	N/A
GPT-2 Medium	SQuADv2	16	5e-5	1e-4	2000	1e-5	160,000	True	N/A
GPT-2 Medium	BoolQ	16	5e-5	1e-4	250	1e-5	12,000	True	N/A
GPT-2 Medium	NarrativeQA	16	5e-5	1e-4	2000	1e-5	170,000	True	N/A
NeMoT	CQA	16	1e-5	N/A	N/A	N/A	N/A	True	True
NeMoT	PIQA	16	1e-5	N/A	N/A	N/A	N/A	True	True
NeMoT	MCTest	16	1e-5	N/A	N/A	N/A	N/A	True	True
NeMoT	WG	16	1e-5	N/A	N/A	N/A	N/A	True	True

TABLE 3.3: Hyperparameters of the conducted experiments in this section. Warm-up loss refers to the loss value the starting loss increases to linearly over "warm-up steps" steps; if it is not utilised then N/A is displayed. The decay loss is the loss value decayed to from the starting loss if no linear warmup, otherwise from the warm-up loss, over "decay steps" steps via cosine decay [51]; if it is not utilised then N/A is displayed. Lm aux loss and vset aux loss refer to two types of auxiliary losses (language modelling auxiliary loss and vanilla set auxiliary loss, respectively): True, False, and N/A refer to the auxiliary loss being used, not used, and it is not applicable, respectively.

The pre-trained NeMoT as described in Section 3.5.1 is utilised as the starting point for fine-tuning on individual datasets. For each dataset, the model is evaluated on the provided validation set and the test set if labels are provided for local evaluation (i.e, not needing a submission to a public leaderboard). Table 3.3 displays the experimental setup for each model and dataset. A rule of thumb is that if cosine decay [51] and a linear warmup is used: the starting loss is 5e-5 and is linearly warmed up over 2000 steps to 1e-4, then decayed via cosine decay to a loss of 1e-5 over 120,000 steps; otherwise a fixed learning rate of 1e-5 is used because of observed training instability with a linear warm-up and cosine decay on that dataset. We utilise Quadro RTX 8000 GPUs, each with the capacity to hold 8 examples in a batch, therefore, the number of GPUs is the batch size divided by 8 in our experiments.

Denote the generated predictions of NeMoT and GPT-2 Medium by O_{pred}, and

that for the vanilla set auxiliary task by O_{vset} (see Figure 3.1). In this section, we optimise the following objective

$$\mathbb{L} = \mathbb{L}_1(O_{\text{pred}}) + 0.2\mathbb{L}_2(O_{\text{pred}}) + 0.5\mathbb{L}_3(O_{\text{vset}}), \tag{3.16}$$

where $\mathbb{L}_1(O_{\text{pred}})$ is the generated loss (e.g., cross entropy) for the current task for the output O_{pred}; $\mathbb{L}_2(O_{\text{pred}})$ is the generated loss for a language modelling objective (i.e., next token prediction) for the output O_{pred}, which is scaled by a factor of 1:5; $\mathbb{L}_3(O_{\text{vset}})$ is the generated loss (e.g., cross entropy) for the current task for the vanilla set's output O_{vset}, which is scaled by a factor of 1:2. GPT-2 Medium does not utilise $\mathbb{L}_3$ in its objective function. Denote $\mathbb{L}_2$ as the language modelling auxiliary loss and $\mathbb{L}_3$ as the vanilla set auxiliary loss.

	Datasets											
Models	SQuADv2 (F_1-score)		RACE-val (Accuracy)			RACE-test (Accuracy)			BoolQ (Accuracy)		NarrativeQA (ROUGE-L)	
	Val	Test	Total	M	H	Total	M	H	Val	Test	Val	Test
GPT-2 Medium♣	0.6608 (ep7)	N/A	63.49 (ep5)	65.95	62.47	62.26	65.04	61.12	76.85 (ep2)	N/A	0.2498 (ep3)	0.2408
NeMoT♣	0.5999 (ep2)	N/A	53.90 (ep4)	56.75	52.71	51.93	54.74	50.77	72.17 (ep2)	N/A	0.3809 (ep2)	0.3750
GPT-2 Medium♠	0.6985 (ep20)	N/A	67.22 (ep12)	69.85	66.13	64.19	66.64	63.18	78.72 (ep17)	N/A	0.2498 (ep3)	0.2408
NeMoT♠	0.6729 (ep16)	N/A	60.02 (ep26)	64.48	58.16	57.94	60.86	56.75	77.13 (ep28)	N/A	0.4147 (ep1)	0.4119
SOTA	N/A	0.9298	N/A	N/A	N/A	90.7	N/A	N/A	N/A	92.4	N/A	0.674

TABLE 3.4: Comparison of the performance of NeMoT versus GPT-2 Medium on four QA datasets, one of each type: multiple-choice, extraction, abstraction, and yes/no. The epoch where the results are taken from is represented by (ep#), where # is a number representing the epoch. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. The epoch with the lowest validation set loss is represented by ♣, while the epoch with the highest metric on the validation set is represented by ♠; the metrics are F_1-score, exact-match accuracy (Accuracy), and ROUGE-L score. The SOTA results (as of March 10th, 2022, excluding ensembles) are reported in [110], [37], [111], and [41] for the test sets of SQuADv2, RACE, BoolQ, and NarrativeQA, respectively.

The results for the experiments conducted comparing NeMoT and GPT-2 Medium are displayed in Table 3.4. We train for at minimum 20 epochs and for some datasets longer, but no more than 30 epochs at maximum. We observe a phenomenon where the validation set's loss converges but the metrics of the dataset (e.g., exact-match accuracy, F_1-score, and ROUGE-L score) are still improving. Rows with ♣ and ♠ represent the results where the lowest validation set loss and highest validation metric score are observed, respectively; the epoch that the results were obtained from is shown as (ep8) for the eighth epoch, for example.

In both models, the epoch where the lowest validation loss was obtained is either the same or lower for NeMoT in comparison to GPT-2 Medium: epoch 2 versus epoch 7, epoch 4 versus epoch 5, epoch 2 versus epoch 2, epoch 2 versus epoch 3, where the former refers to NeMoT's epoch and the latter refers to GPT-2 Medium's epoch. In all datasets excluding NarrativeQA, GPT-2 Medium when choosing the epoch with the lowest validation loss achieves much better performance than that of NeMoT with an absolute difference of 6.09%, 9.59%, 10.33%, and 4.86% for SQuADv2's validation set, RACE's validation set, RACE's test set, and BoolQ's validation set, respectively. Quite strikingly, on NarrativeQA the opposite occurs: NeMoT achieves a ROUGE-L score of 0.3809 and 0.3750 on the validation

and test sets, respectively, while GPT-2 Medium only manages a score of 0.2498 and 0.2408 on the validation and test sets, respectively; an absolute difference of 13.11% and 13.42%, respectively. This phenomena is displayed in Figure 3.3 and discussed in more detail later in this section.

We observe that further training, which in the traditional machine learning sense would be considered overfitting, resulted in the training loss decreasing, validation loss increasing, and surprisingly the metric of the dataset was still improving for both models. We observe which model was better on which datasets are unchanged, however, NeMoT looks to close the gap in performance where it was behind and extend it where it was ahead.

There is not a clear trend if one model achieves the best metric score at a lower or higher epoch than the other. On SQuADv2 and NarrativeQA's validation sets, NeMoT achieves the best metric score at epochs 16 and 1, respectively; lower than epochs 20 and 3 for GPT-2 Medium, respectively. On the RACE and BoolQ validation sets the opposite occurs, i.e, GPT-2 Medium achieves the best metric score at epochs 12 and 17, respectively; lower than epochs 26 and 28 for NeMoT, respectively. We note, however, that the difference may come down to NeMoT being trained for more epochs (GPT-2 Medium is capped at 20 epochs in our experiments due to its larger size); as a result, we cannot extract any meaningful conclusions.

On the SQuADv2 validation set, RACE validation set, RACE test set, and BoolQ validation set we observe GPT-2 Medium still obtaining a better performance with an absolute difference of 2.56%, 7.2%, 6.25%, and 1.59%, respectively. On NarrativeQA, NeMoT achieves a ROUGE-L score of 0.4147 and 0.4119 on the validation and test sets, respectively, while GPT-2 Medium achieves a score of 0.2498 and 0.2408 on the validation and test sets, respectively; an absolute difference of 16.49% and 17.11%, respectively.

While both NeMoT and GPT-2 Medium improve from overfitting, we observe that NeMoT benefits more than GPT-2 Medium by 3.53%, 2.39%, 4.08%, 3.27%, 3.38%, and 3.69% (we subtract the gains of overfitting with NeMoT with that of GPT-2 Medium) for the SQuADv2 validation set, RACE validation set, RACE test set, BoolQ validation set, NarrativeQA validation set, and NarrativeQA test set, respectively. We note that on RACE and BoolQ, NeMoT is trained for more epochs than GPT-2 Medium and may be the cause of the gains on these datasets; however, because the gains are universal over all datasets it might suggest that it is a non-issue.

If we restrict NeMoT on both RACE and BoolQ to only 20 epochs, the best results on RACE's validation set is achieved at epoch 17 with an accuracy of 59.26%, while that for the BoolQ validation set occurs at epoch 18 with an accuracy of 76.94%. Repeating the previous paragraph's calculations for the validation sets of RACE and BoolQ, we achieve a gain by overfitting versus GPT-2 Medium of 1.63% and 3.08%, respectively. Not as much of a gain as before (2.39% and 3.27%, respectively), but still a gain nonetheless, supporting the notion that NeMoT benefits more from overfitting than GPT-2 Medium.

The reason for the divergence of the trend of GPT-2 Medium performing better than NeMoT on NarrativeQA may be due to the fact that, because of the BPE tokenizer splitting whole words into smaller bytes, the tokenized sequence length is often larger than the maximum sequence length of 768, meaning that parts of the passage are cut off, degrading the quality of the dataset. In the other datasets, because of their high quality, overfitting to them is beneficial to the model as it might learn heuristics that while overfit to the training data are useful in terms of generalising to unseen data. With NarrativeQA, as the model quality is degraded by the

shortening of some passages, overfitting does not allow for the learning of useful heuristics that help performance on unseen data, but instead, as in the traditional machine learning sense, degrades performance on unseen data; this is one hypothesised reason for the observed phenomena. Additionally, it is surprising that GPT-2 Medium performs poorly in this scenario; the experiments need to be repeated to determine that it is not due to chance, and then we can dive deeper to determine why this occurs.

When comparing our trained models to the SOTA models there is a large reduction in performance, which is expected because the baseline model and NeMoT are both uni-directional (i.e., they mask future tokens in the attention mechanism), while the SOTA models are bi-directional (i.e., they do not mask any tokens in the attention mechanism), additionally, they consist of many more parameters. The SOTA models should be treated as no more than an indication of what the SOTA performance is on the datasets.

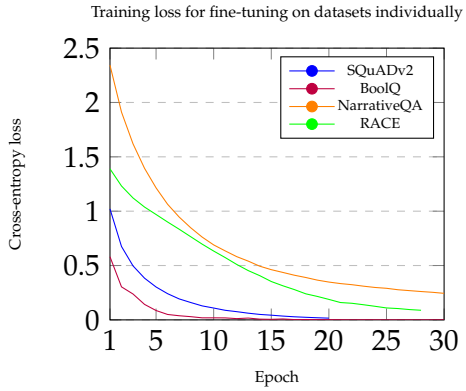
In conclusion, we determine that NeMoT performs worse than GPT-2 Medium in general, with NarrativeQA being an exception that needs to be looked into further. However, when overfitting and choosing the epoch with the best metric score on the validation set, the performance is relatively small. Given our model consists of approximately 84 million fewer parameters, we extract a positive signal from NeMoT and the neuromodulatory mechanism, where if we matched the number of parameters we might expect similar or slightly better performance. Additionally, our implementation of the neuromodulatory mechanism is quite primitive and the optimal hyperparameters have not yet been determined. It is not unreasonable to think that this may have a moderate impact on performance. However, we do restrain from stating that our conclusions are definite — as we would like to repeat the experiments multiple times to generate confidence intervals — but instead, an initial positive signal that needs to be explored further.

The apparent benefits of overfitting

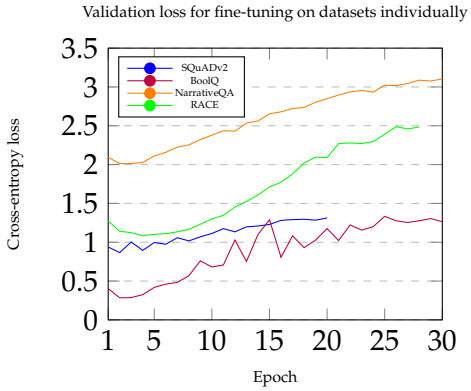
The training set losses, validation set losses, and validation set metric results are shown in Figure 3.3 for both NeMoT and GPT-2 Medium when fine-tuning on RACE, SQuADv2, BoolQ, and NarrativeQA. For NeMoT, plots (A), (B), and (C) represent the training loss, validation loss, and validation metrics, respectively. We observe the trend via plot (B) that the validation losses converge quite quickly (epochs 2, 4, 2, and 2 for SQuADv2, RACE, BoolQ, and NarrativeQA, respectively) and start to, in the traditional machine learning sense, overfit; in normal circumstances, the training would halt. However, plot (C) illustrates that on SQuADv2, RACE, and BoolQ's validation sets that further training in what would otherwise be called overfitting, is still improving the metrics of their respected datasets.

For SQuADv2 we observe that at 20 epochs the model's F_1-score is either still improving slightly with further training, or has converged. We observe that RACE is still improving in the later epochs, with the best accuracy observed at epoch 26; epochs 27 and 28 are marginally lower and may suggest a convergence. BoolQ reaches a maximum accuracy at epoch 28, possibly having converged.

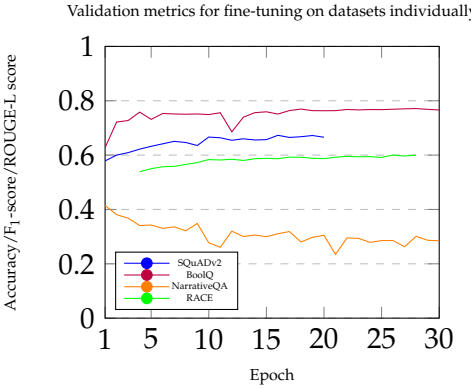
NarrativeQA is the only dataset where NeMoT acts as we would expect, i.e., when the validation loss converges and starts to increase the ROUGE-L score degrades with it. The lowest validation loss occurs at epoch 2, while the best ROUGE-L score occurs at epoch 1. The validation metric generally degrades at each passing epoch, contradictory to the other datasets. The reason for such is likely because some inputs to the model have a sequence length — after tokenization, into the smallest,



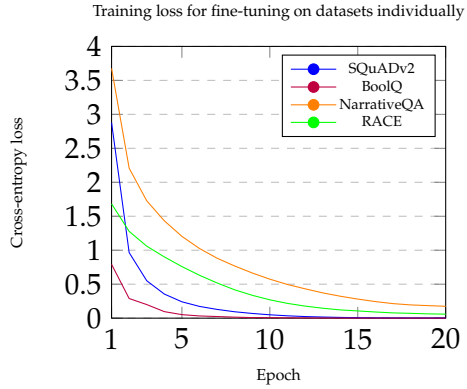
(A) NeMoT training loss



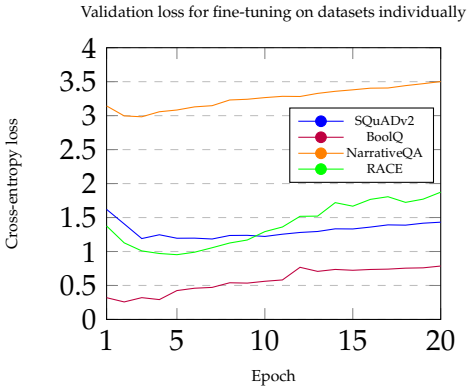
(B) NeMoT validation loss



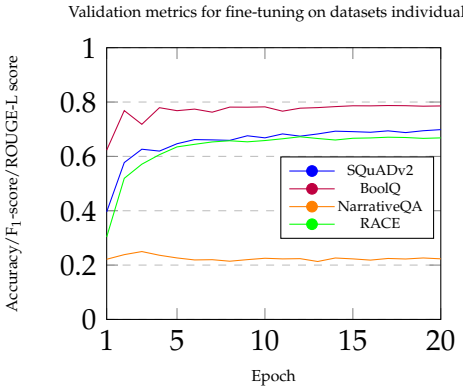
(C) NeMoT validation metrics



(D) Baseline training loss



(E) Baseline validation loss



(F) Baseline validation metrics

FIGURE 3.3: Plots portraying the observed overfitting phenomena where the training set loss decreases, the validation set loss increases, and the validation set metric improves.

characters — much larger than 768, the maximum supported sequence length. Thus, to make the input fit into the model we remove tokens from the beginning of the passage of the input to match the desired sequence length, degrading the quality of the dataset.

Plot (D), (E), and (F) represent the baseline model's training loss, validation loss, and validation metrics, respectively. We observe similar phenomena in the baseline model to that of NeMoT: the same conclusions for NarrativeQA hold. We observe that at epoch 20, performance on SQuADv2 is either still improved slightly with further training, or to have converged; RACE is observed to have converged at epoch 12, with no improvement at later epochs; BoolQ looks to have converged at epoch 17 with no improvement at later epochs. Plot (E) showcases the longer convergence time of GPT-2 Medium: epochs 7, 5, 2, and 3 for SQuADv2, RACE, BoolQ, and NarrativeQA, respectively.

One theory as to why we observed the divergence between the validation loss and validation metric is the quality and possibly size of the datasets, where overfitting allows better heuristics to be learned by the model, hence, better performance on unseen datasets, contrary to prior belief. Supporting evidence for such a claim is the degraded version of NarrativeQA, where the beginning of the passage is cut to match the maximum sequence length; some samples may cut out the part essential to the generation of the answer. Overfitting to lower quality datasets, like NarrativeQA in our experiments, does not allow useful information to be extracted with further training, like that observed with RACE, SQuADv2, and BoolQ.

Additional experiments for future sections

In Sections 3.5.4 and 3.5.5 we conduct experiments on four MQA datasets: CommonsenseQA (CQA), Physical Interaction QA (PIQA), MCTest, and WinoGrande (WG). Here, we provide a baseline NeMoT model to act as a comparison to in future sections; the experimental setup is shown in Table 3.3 and the experiment results are illustrated in Table 3.5.

The overfitting phenomena is observed in Table 3.4, with each dataset benefiting from further training after the validation loss converges and starts to increase. In CQA, we observe the lowest validation loss occurring at epoch 8 and the highest validation accuracy occurring at epoch 15. Epoch 8's validation accuracy is 29.65%, while epoch 15's validation accuracy is 33.74%, a difference of 4.09%. In PIQA, we observe the lowest validation loss occurring at epoch 6 and the highest validation accuracy occurring at epoch 20. Epoch 6's validation accuracy is 53.48%, only 3.48% better than random, while epoch 20's validation accuracy is 59.63%, a difference of 6.15%. In MCTest, we observe the biggest difference between the lowest validation loss's accuracy and the global best validation's accuracy. The lowest validation loss occurs at epoch 6 and the highest validation accuracy occurs at epoch 19. Epoch 6's validation accuracy is 29.69%, while epoch 19's validation accuracy is 43.75%, a difference of 14.06%. For the test set, epochs 6 and 19 result in an accuracy of 25.71% and 44.05%, respectively; a difference of 18.34%. In WG, we observe the lowest validation loss occurring at epoch 2 and the highest validation accuracy occurring at epoch 14. Epoch 2's validation accuracy is 51.07%, 1.07% better than random, while epoch 14's validation accuracy is 59.98%, a difference of 8.91%.

The overfitting phenomena observed in Table 3.4, where performance increases with overfitting, carries over and is observed for all datasets. All reported results are better than randomly selecting a label, but are nowhere near the SOTA performance

	Datasets							
Models	CQA (Accuracy)		PIQA (Accuracy)		MCTest (Accuracy)		WG (Accuracy)	
	Val	Test	Val	Test	Val	Test	Val	Test
NeMoT♣	29.65 (ep8)	N/A	53.48 (ep6)	N/A	29.69 (ep6)	25.71	51.07 (ep2)	N/A
NeMoT♠	33.74 (ep15)	N/A	59.63 (ep20)	N/A	43.75 (ep19)	44.05	59.98 (ep14)	N/A
Random	0.2	0.2	0.5	0.5	0.25	0.25	0.5	0.5
SOTA	N/A	80.7	N/A	90.13	95.6	71.0†	N/A	88.29

TABLE 3.5: Performance of NeMoT on four multiple-choice QA datasets. The epoch where the results are taken from is represented by (ep#), where # is a number representing the epoch. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. The random row is the performance we would expect if we randomly selected an answer. The epoch with the lowest validation set loss is represented by ♣, while the epoch with the highest metric on the validation set is represented by ♠; every dataset's metric is exact-match accuracy (Accuracy). The SOTA results (as of March 10th, 2022, excluding ensembles) are reported in [100], [52], and [52] for the test sets of CQA, PIQA, and WG, respectively. †We note that for MCTest the SOTA results for the test set is 71% [89], while the best on the validation set is much higher at 95.6% [41], but they do not report test results. It is not far fetched to expect the model with 95.6% accuracy on the validation set to perform similarly on the test set, thus, we consider it the SOTA model.

as expected because of the smaller number of parameters and the fact that the model is uni-directional, while the SOTA models are bi-directional.

3.5.3 General fine-tuning

We are mainly interested in if NeMoT results in an improved generalisation, i.e., an improved performance in the GQA setting; we test such in this section. We use the training procedure outlined in Section 3.3, including the seed datasets: RACE [45], SQuADv2 [69], NarrativeQA [44], BoolQ [15], ARC [16], OBQA [59], and MCTest [72]. ARC and OBQA do not utilise any information retrieval, hence, the model needs to have relevant information stored internally in its parameters. We test a model's generalisation capabilities in a zero-shot setting on non-seed datasets (i.e., datasets whose training set has not been trained on): Quoref [21], ROPES [48], DROP [25], CQA [88], WG [73], PIQA [7], SIQA [74], Quoref-CS [21, 28], ROPES-CS [28, 48], DROP-CS [25, 28], and BoolQ-CS [15, 28].

We train two models: $\text{NeMoT}_{\text{general}}$ and $\text{GPT-2 Medium}_{\text{general}}$ on the seed datasets, representing the generally trained version of NeMoT and GPT-2 Medium, respectively; both models are uni-directional. In both instances a batch size of 2 is utilised with one Quadro RTX 8000 GPU; the loss function is categorical cross entropy; the learning rate linearly warms up from 5e-5 to 1e-4 over 2000 steps, then decays from 1e-4 to 1e-5 over 400,000 steps via cosine decay [51]; the ADAM optimizer [43] with default parameters is utilised.

Denote the generated predictions of $\text{NeMoT}_{\text{general}}$ and $\text{GPT-2 Medium}_{\text{general}}$ by O_{pred}, and that for the vanilla set auxiliary task by O_{vset} (see Figure 3.1). In this

section, we optimise the following objective

$$\mathbb{L} = \mathbb{L}_1(O_{\text{pred}}) + 0.2\mathbb{L}_2(O_{\text{pred}}) + 0.5\mathbb{L}_3(O_{\text{vset}}), \tag{3.17}$$

where $\mathbb{L}_1(O_{\text{pred}})$ is the generated loss (e.g., cross entropy) for the current task for the output O_{pred}; $\mathbb{L}_2(O_{\text{pred}})$ is the generated loss for a language modelling objective (i.e., next token prediction) for the output O_{pred}, which is scaled by a factor of 1:5; $\mathbb{L}_3(O_{\text{vset}})$ is the generated loss (e.g., cross entropy) for the current task for the vanilla set's output O_{vset}, which is scaled by a factor of 1:2. GPT-2 Medium$_{\text{general}}$ does not utilise $\mathbb{L}_3$ in its objective function. Denote $\mathbb{L}_2$ as the language modelling auxiliary loss and $\mathbb{L}_3$ as the vanilla set auxiliary loss.

In NeMoT$_{\text{general}}$, gradient flow during back-propagation is not restricted at any point. Two auxiliary losses are used: the language modelling auxiliary loss and the vanilla set auxiliary loss. In GPT-2-Medium$_{\text{general}}$, only a language modelling auxiliary loss is utilised, i.e., $\mathbb{L}_2$. In both cases the experimental results are reported for the model trained for 400,000 iterations.

Model	Parameters	Layers	Dimension	FFN dimension	Heads	Pre-training data
NeMoT	280M	30†	768	3072	12	20GB*
GPT-2 Medium	345M	24	1024	4096	16	40GB

TABLE 3.6: A comparison of NeMoT and GPT-2 Medium: parameters, layers, dimension, ffn dimension, heads, and pre-training data represent the number of parameters, the number of layers, the dimension of the model, the dimension of the point-wise feed-forward network, the number of heads in multi-head attention, and the amount of data trained on during pre-training in gigabytes (GB), respectively. "M" represents millions. *Not including the 40 GB of data that GPT-2 Small, the acting vanilla set was pre-trained on. †Includes two output set blocks, each of 3 layers, that will be run in parallel, hence, in practice, there are 27 layers in one traversal through the model, but the last three can change depending on what block in the output set is traversed.

NeMoT$_{\text{general}}$ is initialised to the pre-trained model produced in Section 3.5.1. In the general version of the model, we utilise two parallel Transformer blocks in the output set named *mqa* and *gqa*, representing multiple-choice question answering and generate question answer — the auto-regressive generation of a sequence of text to answer the question — respectively. They are both initialised to the weights of the *lm* Transformer block in the output set. If the auxiliary task token is <*mqa*> then we traverse the mqa Transformer block; if it is <*gqa*> then we traverse the gqa Transformer block. The GPT-2-Medium$_{\text{general}}$ pre-trained model from Hugging Face is utilised, which excludes the last fully connected layer; we initialise randomly a fully connected layer of the same dimensions to replace it.

For a comparison of the two models see Table 3.6. Notable differences between the two models are the approximately 65 million more parameters, higher dimension, higher point-wise feed-forward network dimension, and more heads for GPT-2 Medium. Compared to the models in Section 3.5.2 we see that NeMoT closes the gap in terms of the number of parameters and explicitly encodes modularity into the output set, allowing multiple-choice QA tasks to traverse a different set of layers than other QA tasks.

Models	Datasets										
	SQuADv2 (F_1-score)		RACE-val (Accuracy)			RACE-test (Accuracy)			ARC-val (Accuracy)		
	Val	Test	Total	M	H	Total	M	H	Total	Hard	Easy
GPT-2 Medium$_{\text{general}}$	0.5405	N/A	52.57	55.64	51.29	51.07	55.92	49.09	36.59	33.78	38.07
NeMoT$_{\text{general}}$	0.5272	N/A	48.43	51.74	47.06	48.43	50.97	45.54	35.90	33.78	37.02
UnifiedQA$_{\text{Base}}$	0.781	N/A	N/A	N/A	N/A	70.3	N/A	N/A	N/A	N/A	N/A
UnifiedQA$_{\text{11B}}$	0.898	N/A	N/A	N/A	N/A	88.6	N/A	N/A	N/A	N/A	N/A
SOTA	N/A	0.9298	N/A	N/A	N/A	90.7	N/A	N/A	N/A	N/A	N/A

Models	Datasets										
	ARC-test (Accuracy)			OBQA (Accuracy)		MCTest (Accuracy)		BoolQ (Accuracy)		NarrativeQA (ROUGE-L)	
	Total	Hard	Easy	Val	Test	Val	Test	Val	Test	Val	Test
GPT-2 Medium$_{\text{general}}$	37.88	33.62	39.98	47.60	45.60	57.19	57.50	74.40	N/A	0.2608	0.2611
NeMoT$_{\text{general}}$	38.53	32.59	41.46	48.40	49.80	51.25	56.19	74.95	N/A	0.2839	0.2791
UnifiedQA$_{\text{Base}}$	N/A	45.7	58.5	N/A	59.4	86.9	N/A	N/A	82.5	N/A	0.604
UnifiedQA$_{\text{11B}}$	N/A	77.8	87.2	N/A	86.4	95.6	N/A	N/A	90.3	N/A	0.674
SOTA	86.52	N/A	N/A	N/A	87.4	95.6	71.0†	N/A	92.4	N/A	0.674

TABLE 3.7: In-domain results on the seed datasets; comparison of NeMoT and GPT-2 Medium. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. The metrics are F_1-score, exact-match accuracy (Accuracy), and ROUGE-L score. UnifiedQA's results are obtained from [41], where the best result from either version one or two is reported for each dataset. The SOTA results (as of March 10th, 2022, excluding ensembles) are reported in [110], [37], [111], [95], [111], and [41] for the test sets of SQuADv2, RACE, ARC, OBQA, BoolQ, and NarrativeQA, respectively. †We note that for MCTest the SOTA results for the test set is 71% [89], while the best on the validation set is much higher at 95.6% [41], but they do not report test results. It is not far fetched to expect the model with 95.6% accuracy on the validation set to perform similarly on the test set, thus, we consider it the SOTA model.

We report results for both models after 400,000 iterations of training in Tables 3.7 and 3.8, representing the in-domain results and the out-of-domain results, respectively. In-domain results are obtained on the seed dataset's evaluation sets and test sets if the labels are provided for local evaluation. Out-of-domain results are obtained on the non-seed datasets' evaluation sets. The out-of-domain datasets are often quite different — such as DROP, which includes arithmetic operations — to the seed datasets and provide a good indicator of generally acquired knowledge.

When training exclusively on a dataset we previously concluded that GPT-2 Medium performs better than NeMoT, with only NarrativeQA bucking the trend. On the in-domain datasets, GPT-2 Medium performs better on the SQuADv2 validation set, RACE validation and test sets, ARC validation set, and MCTest validation and test sets; NeMoT performs better on the ARC test set, OBQA validation and test sets, BoolQ validation set, and NarrativeQA validation and test sets. Overall, both models seem to be mostly on par with one another, a positive sign for NeMoT which was generally worse than GPT-2 Medium when fine-tuning on individual datasets, which was likely largely due to the reduced number of parameters of NeMoT in comparison.

In comparison to fine-tuning on individual datasets, the general versions of NeMoT and GPT-2 Medium both perform worse on all four datasets. On

SQuADv2's validation set, the general versions perform 0.158 and 0.1457 worse for GPT-2 Medium and NeMoT, respectively. On RACE's validation set, the general versions perform 14.64% and 11.59% worse for GPT-2 Medium and NeMoT, respectively. On RACE's test set, the general versions perform 13.12% and 9.51% worse for GPT-2 Medium and NeMoT, respectively. On BoolQ's validation set, the general versions perform 4.32% and 2.18% worse for GPT-2 Medium and NeMoT, respectively. On NarrativeQA's validation set, the general versions perform 0.011 better and 0.1308 worse for GPT-2 Medium and NeMoT, respectively. Lastly, on NarrativeQA's test set, the general versions perform 0.0203 better and 0.1328 worse for GPT-2 Medium and NeMoT, respectively.

Reasons for such may include the sharing of a model's parameters with multiple datasets, however, we have reason to expect this may help the model in some instances [42]. Additionally, more than 400,000 iterations may be needed to achieve better performance, especially since in Section 3.5.2 we found overfitting to be beneficial.

UnifiedQA is a model that was trained similarly to ours in terms of the general training procedure, with the main difference being that the model is an encoder-decoder Transformer, allowing for the processing of the input in a bi-directional manner, giving it a natural advantage over NeMoT and GPT-2 Medium. It serves as a good comparison to see the difference in performance when each token position is allowed to attend to all other token positions. On the in-domain datasets, NeMoT and the baseline always perform worse than UnifiedQA$_{\text{Base}}$ — which consists of only 220 million parameters, less than NeMoT and GPT-2 Medium — and often by a decent margin. We should not look too much into the difference but should note the degradation of performance in uni-directional models.

Therefore, for the in-domain datasets, we see the performance of both models is quite similar, with NeMoT performing better in some instances and GPT-2 Medium in others. Performance is worse for the generally trained models in comparison to training on a dataset individually for RACE, SQuADv2, and BoolQ for both models; better on NarrativeQA for the baseline and worse for NeMoT.

The out-of-domain datasets, as displayed in Table 3.8, generally showcase a poor performance for both NeMoT and GPT-2 Medium. For the extractive datasets, we see NeMoT perform slightly better on Quoref, Quoref-CS, and ROPES-CS by a margin of 0.0023, 0.0046, and 0.011, respectively; GPT-2 Medium performs better only on ROPES by a margin of 0.0559, however, it performs worse on the contrast set version of the dataset. For the abstraction datasets, they perform similarly with identical performance on DROP, but slightly better for GPT-2 Medium on the contrast set version of the dataset by a small margin of 0.001. For the yes/no dataset BoolQ-CS, GPT-2 Medium performs better than NeMoT by a margin of 2.73%. For multiple-choice datasets, NeMoT performs better on WG, PIQA, and SIQA by a margin of 19.58%, 21.33%, and 3.33% respectively; it is important to note that these results are worse than random: 0.5, 0.5, and $0.\overline{3}$ for WG, PIQA, and SIQA, respectively. GPT-2 Medium performs better than NeMoT on CQA by a margin of 1.48%. We note that CQA consists of five answer options, while during training the model was only trained to generate an answer for at most four answer options (i.e., it has never seen the first answer option's label); we note that this will harm performance. Additionally, note that on the datasets with only two answer options we could just flip the results to achieve a much higher performance; however, some incorrect samples are because the model does not output an answer option to begin with, so it is not so simple. We do question if the model has actually learned anything about these datasets, or if it is entirely due to chance.

Models	Datasets								
	Quoref (F_1-score)		Quoref-CS (F_1-score)	ROPES (F_1-score)		ROPES-CS (F_1-score)	DROP (ROUGE-L)		DROP-CS (ROUGE-L)
	Val	Test	Test	Val	Test	Test	Val	Test	Test
GPT-2 $\text{Medium}_{\text{general}}$	0.1200	N/A	0.1708	0.0940	N/A	0.0635	0.0183	N/A	0.0153
$\text{NeMoT}_{\text{general}}$	0.1223	N/A	0.1754	0.0381	N/A	0.0745	0.0183	N/A	0.0143
$\text{UnifiedQAv1}_{\text{Base}}$	40.0		38.5	33.9		22.8	19.7		23.7
$\text{UnifiedQAv1}_{\text{11B}}$	63.5		55.3	67.0		45.5	32.5		40.1
SOTA	N/A	0.8670	0.554	N/A	0.8025	0.455	N/A	0.9010*	0.542

Models	Datasets								
	BoolQ-CS (Accuracy)	CQA (Accuracy)		WG (Accuracy)		PIQA (Accuracy)		SIQA (Accuracy)	
	Test	Val	Test	Val	Test	Val	Test	Val	Test
GPT-2 $\text{Medium}_{\text{general}}$	58.42	21.87	N/A	17.36	N/A	10.39	N/A	24.92	N/A
$\text{NeMoT}_{\text{general}}$	55.69	20.39	N/A	36.94	N/A	31.72	N/A	28.25	N/A
$\text{UnifiedQAv1}_{\text{Base}}$	61.9	45.0		N/A		N/A		N/A	
$\text{UnifiedQAv1}_{\text{11B}}$	80.4	76.2		N/A		N/A		N/A	
SOTA	80.4	N/A	83.3	N/A	88.29	N/A	90.13	N/A	83.15

TABLE 3.8: Out-of-domain results on the non-seed datasets; comparison of NeMoT and GPT-2 Medium. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. The metrics are F_1-score, exact-match accuracy (Accuracy), and ROUGE-L score. The SOTA results (as of March 10th, 2022, excluding ensembles) are reported in [104], [42], [42], [42], [12, 102], [42], [42], [100] [52], [52], and [52] for the test sets of Quoref, Quoref-CS, ROPES, ROPES-CS, DROP, DROP-CS, BoolQ-CS, CQA, WG, PIQA, and SIQA, respectively. UnifiedQA's results are taken from the first version of the model [42]; the paper does not state if the results came from the validation set or the test set, therefore, the results cover both the validation and test cells, portraying the ambiguity. *Note that the F_1-score is reported here.

As with the in-domain datasets, $\text{UnifiedQAv1}_{\text{Base}}$ performs much better by a margin of around 20% on most datasets, with BoolQ-CS being a notable exception with a difference of only 3.48% to the baseline. $\text{UnifiedQAv1}_{\text{Base}}$ serves as a good comparison to see the impact that NeMoT and GPT-2 Medium both being uni-directional has on performance. However, we note the reduction in performance may not be only because the model is uni-directional and may be due to other factors; we refrain from saying that being uni-directional is the definitive cause, but it is a likely cause.

In conclusion, the out-of-domain datasets show that NeMoT performs better than GPT-2 Medium in all but four instances: ROPES, DROP-CS, BoolQ-CS, and CQA. The biggest differences in NeMoT's favour occur in WG and PIQA (19.58% and 21.33% respectively); the biggest for GPT-2 Medium occur in BoolQ-CS and ROPES (2.73% and 0.0559 respectively). We note that further experiments are needed to confirm the results and that they are not due to chance (i.e., construct confidence intervals).

Overall, the performance of NeMoT and GPT-2 Medium is quite similar on the in-domain datasets, with some datasets favouring NeMoT and others GPT-2 Medium; the out-of-domain datasets tend to favour NeMoT, suggesting an improved generalisation. Given the circumstances — including the simple implementation of the neuromodulatory mechanism and the smaller number of parameters it has in comparison to GPT-2 Medium — we extract a positive signal from NeMoT

and the neuromodulatory mechanism in regards to a potential improvement in generalisation capabilities. However, the conclusions drawn in this section are not conclusive and further experiments are needed to definitively conclude such, including the contribution of the manually introduced modularity in the output set via the parallel blocks of Transformers.

3.5.4 Fine-tuning the generally trained model on individual datasets

UnifiedQA, a generally trained model on multiple QA datasets and formats, when fine-tuning on individual datasets, resulted in a better performance than fine-tuning on a vanilla language model [42]. In this section, we test if the same holds with NeMoT on MQA datasets. Specifically, does utilising $\text{NeMoT}_{\text{general}}$ as a starting point for fine-tuning on individual datasets result in a better performance than starting with the pre-trained NeMoT ($\text{NeMoT}_{\text{pre-trained}}$)?

We utilise four MQA datasets: CQA [88], PIQA [7], MCTest [72], and WG [73]. The experiments are set up identically to the results obtained in Table 3.5, with the only difference being that $\text{NeMoT}_{\text{general}}$ is the starting model preluding fine-tuning; $\text{NeMoT}_{\text{pre-trained}}$ represents the results obtained in the Table 3.5. This section's results are illustrated in Table 3.9.

	Datasets							
Models	CQA (Accuracy)		PIQA (Accuracy)		MCTest (Accuracy)		WG (Accuracy)	
	Val	Test	Val	Test	Val	Test	Val	Test
$\text{NeMoT}_{\text{pre-trained}}$♣	29.65 (ep8)	N/A	53.48 (ep6)	N/A	29.69 (ep6)	25.71	51.07 (ep2)	N/A
$\text{NeMoT}_{\text{general}}$♣	25.88 (ep6)	N/A	52.83 (ep2)	N/A	51.25 (ep1)	56.07 (ep1)	52.57 (ep2)	N/A
$\text{NeMoT}_{\text{pre-trained}}$♠	33.74 (ep15)	N/A	59.63 (ep20)	N/A	43.75 (ep19)	44.05	59.98 (ep14)	N/A
$\text{NeMoT}_{\text{general}}$♠	31.29 (ep17)	N/A	55.50 (ep14)	N/A	51.25 (ep1/ep2)	56.07 (ep2)	57.22 (ep18)	N/A
SOTA	N/A	83.3	N/A	90.13	95.0	71.0†	N/A	88.29

TABLE 3.9: Comparing the performance of the generally trained NeMoT and pre-trained NeMoT, fine-tuned on four multiple-choice QA datasets. The epoch where the results are taken from is represented by (ep#), where # is a number representing the epoch. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. The epochs with the lowest validation set loss are represented by ♣, while the epochs with the highest metric on the validation set are represented by ♠; every dataset's metric is exact-match accuracy (Accuracy). The SOTA results (as of March 10th, 2022, excluding ensembles) are reported in [100], [52], and [52] for the test sets of CQA, PIQA, and WG, respectively. †We note that for MCTest the SOTA results for the test set is 71% [89], while the best on the validation set is much higher at 95.6% [41], but they do not report test results. It is not far fetched to expect the model with 95.6% accuracy on the validation set to perform similarly on the test set, thus, we consider it the SOTA model.

Surprisingly, contrary to the results obtained in [42], we observe that for the most part $\text{NeMoT}_{\text{pre-trained}}$ performs better as a starting point than $\text{NeMoT}_{\text{general}}$. $\text{NeMoT}_{\text{pre-trained}}$ performs better on CQA and PIQA for both the lowest validation loss's epoch and the highest validation metric's epoch; for WG, it performs better only on the highest validation metric's epoch.

$\text{NeMoT}_{\text{general}}$ performs better as a starting point on WG for the lowest validation loss's epoch; on MCTest for both the lowest validation loss's epoch and the

highest validation metric's epoch. Between both models, the largest difference in performance is observed in MCTest in $\text{NeMoT}_{\text{general}}$'s favour. The difference may be explained by the fact that MCTest was included in the seed datasets in $\text{NeMoT}_{\text{general}}$ and was previously trained on, while the other datasets were not.

Overall, fine-tuning $\text{NeMoT}_{\text{pre-trained}}$ outperforms that of the generally trained model on all datasets except MCTest when "overfitting". The anomaly of MCTest might be explained by the fact that MCTest was included in the seed datasets and trained on in $\text{NeMoT}_{\text{general}}$; all other datasets were not included in the seed datasets. This may be an artefact of utilising uni-directional models versus bi-directional models, or that more parameters are needed to see the benefits observed in [42], but further experiments are needed to test such claims.

3.5.5 Reading strategies

Reading strategies are correlated with an improved reading proficiency and reading comprehension in humans [61, 71]; they have been shown to improve performance in QA [86, 109]. We test the contribution of two reading strategies introduced into NeMoT on four MQA datasets: CommonsenseQA (CQA) [88], Physical Interaction QA (PIQA) [7], MCTest [72], and WG [73]. The two reading strategies are answer-option interaction (AOI) and highlighting. The experiments are set up identically to the results obtained in Table 3.5 in Section 3.5.2, with the only different being the models we train: $\text{NeMoT}_{\text{general}}$, $\text{NeMoT}_{\text{aoint}}$, and $\text{NeMoT}_{\text{high}}$, representing the generally trained NeMoT, the generally trained NeMoT with the AOI reading strategy, and the generally trained NeMoT with the highlighting reading strategy, respectively. $\text{NeMoT}_{\text{aoint}}$ and $\text{NeMoT}_{\text{high}}$ are both initialised to $\text{NeMoT}_{\text{general}}$'s parameters and the newly introduced parameters by the reading strategies are randomly initialised via the Xavier uniform initializer [29]. For a description of both reading strategies see Section 3.4.3.

Models	Datasets							
	CQA (Accuracy)		PIQA (Accuracy)		MCTest (Accuracy)		WG (Accuracy)	
	Val	Test	Val	Test	Val	Test	Val	Test
$\text{NeMoT}_{\text{general}}$	31.29	N/A	55.50	N/A	51.25	56.07	57.22	N/A
$\text{NeMoT}_{\text{aoint}}$	31.94	N/A	57.83	N/A	51.56	54.40	57.77	N/A
$\text{NeMoT}_{\text{high}}$	31.29	N/A	57.67	N/A	53.13	54.29	58.48	N/A

TABLE 3.10: Performance of two reading strategies integrated with NeMoT on four multiple-choice QA datasets. The epoch where the results are taken from is represented by (ep#), where # is a number representing the epoch. Cells where we report no results are depicted by N/A; all results are rounded to four significant figures; we only report results on the test sets if the labels are provided for local evaluation. $\text{NeMoT}_{\text{general}}$ is the baseline model and every datasets' metric is exact-match accuracy (Accuracy).

We observe that $\text{NeMoT}_{\text{aoint}}$ outperforms $\text{NeMoT}_{\text{general}}$ by 0.65% on CQA's validation set, 2.33% on PIQA's validation set, 0.31% on MCTest's validation set, and 0.55% on WG's validation set. However, it performs worse on MCTest's test set by a margin of 1.67%.

We observe similar trends with $\text{NeMoT}_{\text{high}}$, with minor exceptions. It performs better than $\text{NeMoT}_{\text{general}}$ by 2.17% on PIQA's validation set, 1.88% on MCTest's validation set, and 1.26% on WG's validation set; performance is identical on CQA's validation set. However, it performs worse on MCTest's test set by a margin of 1.78%.

When comparing the two reading strategies to one another we find that $\text{NeMoT}_{\text{aoint}}$ performs better on CQA's validation set by 0.65%, PIQA's validation set by 0.16%, and MCTest's test set by 0.11%. $\text{NeMoT}_{\text{high}}$ performs better on MCTest and WG's validation sets by 1.57% and 0.71%, respectively.

In conclusion, we see that both reading strategies improve performance in general versus $\text{NeMoT}_{\text{general}}$ with the exception of MCTest's test set. When comparing the reading strategies to one another we find that they perform better on different datasets; given the limited scope of these experiments, we cannot determine if one is better than the other. To definitively claim that the reading strategies are an improvement over $\text{NeMoT}_{\text{general}}$ we would like to run the experiments from scratch multiple times to generate confidence intervals and expand the experiments to include more MQA datasets. We note that if we applied the reading strategies directly after the embedding layer we would expect further improvements.

3.6 Discussion

We have tested the capabilities of NeMoT on a variety of tasks: zero-shot language modelling via a pre-trained language model; fine-tuning the pre-trained language model on datasets individually; fine-tuning in a GQA setting, where we test its ability to generalise to unseen datasets; and testing the capabilities of two simple reading strategies on MQA datasets.

The observed overfitting phenomena, where the validation loss increases but the validation metric improves, is described in [30] as a calibration issue of the true correctness likelihood. i.e., the model is overconfident in predictions that it gets incorrect, or underconfident in predictions that it gets correct; it is a problem in domains where a model accurately displaying its confidence in an answer is essential, such as medical diagnosis, for example. They label it as a common phenomenon with modern neural networks [30]. Using this logic, in our case our model is either becoming more confident in incorrect predictions, a possible reason for the divergence in the validation's loss and metric, or is becoming less confident in correct predictions, but not enough to choose the incorrect answer, all while it is predicting more examples correct.

NeMoT and GPT-2 Medium, which are both uni-directional Transformers, have the same calibration issues, possibly suggesting that it is problem with uni-directional models. Experiments to see if the same occurs in bi-directional Transformers (i.e., BERT) will determine if it is a common phenomenon in Transformers, or potentially just a problem with uni-directional Transformers. Alternatively, it could be a phenomenon with the text-to-text framework utilised by both models.

Our pre-trained language model, while still improving when we stopped training on C4, is worse than GPT-2 Medium in zero-shot language modelling (see Table 3.1). Our model consists of fewer parameters than GPT-2 Medium but has been trained on more data, especially when we include the data used to pre-train the vanilla set (i.e., the 40 GB GPT-2 Small was previously trained on). The difference in performance might be because of the different pre-training datasets utilised as NeMoT performs better on PTB, but not on the other three datasets, than GPT-2

Medium; also, the fact that we traversed only a small subset of C4's data may or may not play a role. We can only postulate the impact more training on C4 would have for NeMoT, but the same goes for the GPT-2 models.

NeMoT lags slightly in performance when training on individual datasets in comparison to GPT-2 Medium, which is expected because of the 84 million more parameters that GPT-2 Medium has. We postulated that the neuromodulatory mechanism in NeMoT would be better suited for GQA because it would allow for more complex rules to be learned, that when coupled with an environment that encourages generalisation (i.e., multi-task learning), would result in a better performance in GQA. Our experiments in GQA support such a notion, but more experiments need to be conducted to make a definitive conclusion.

We observed that both reading strategies performed better than the baseline and that we could not conclude that one was better than the other in general. While both did improve performance, it may not be for the intended reasons. The resulting improvement by AOI may be due to the fact that it simply allows all answer options positions to see one another, which would otherwise not be the case because future tokens are masked (i.e., because the model is uni-directional). Seeing the same increase in performance in a model where future tokens are not masked (i.e., in a bi-directional model) would remedy the concern. Additionally, for both reading strategies, we have no proof that the improved performance is because of the reading strategies themselves and not because of the additional parameters introduced by them. Before concluding that the improved performance is due to the reading strategies themselves, we need to account for the previously mentioned concerns.

Chapter 4

Conclusion

In this thesis, we set out to improve the generalisation capabilities of QA models, or more specifically the Transformer. We focus on extending the Transformer via the entwinement of neuromodulation. We hypothesise that the addition of neuromodulation, when coupled with an environment that encourages it (e.g., multi-task and multi-format learning), will result in better generalisation capabilities. We introduce the Neuromodulated Transformer (NeMoT), a new Transformer architecture. We test NeMoT's QA capabilities in QA on individual datasets and in a GQA setting, where it is required to generalise to out-of-domain datasets that it has not encountered before. Additionally, as a secondary objective in this thesis, we integrate and extend reading strategies with NeMoT and measure their performance on MQA datasets.

In this chapter, we feature the achievements of our thesis, highlight all limitations and caveats with the concluded results, and list possible pathways for further study.

4.1 Achievements

The following bullet points highlight what was achieved in this thesis:

- We introduced the Neuromodulated Transformer (NeMoT), a simple entwinement of neuromodulation with the Transformer architecture.
- We showcase that there is potential for neuromodulation in the Transformer architecture in GQA, especially given the observed improvement versus a baseline in GQA on out-of-domain datasets.
- We modify two simple reading strategy implementations (AOI and highlighting) and observe an improved performance in MQA in comparison to if they were not included.

4.2 Research questions

The following bullet points state our research questions and answer them:

- Does the entwinement of neuromodulation with the Transformer architecture, when coupled with an environment that encourages it, allow for better generalisation in QA? Our experiments suggest that neuromodulation, when entwined with the Transformer architecture, results in better generalisation, but the results are not conclusive. Further experimentation is needed to more confidently determine such.

- Can the utilisation of reading strategies improve performance in MQA? Yes, our implementations of reading strategies (i.e., answer option interaction and highlighting) result in an improved performance over a baseline model with no reading strategies.
- Does the entwinement of neuromodulation with the Transformer architecture improve its QA capabilities on individual datasets? We extract no meaningful conclusions about QA on individual datasets, largely due to the fact that the baseline model consists of 84 million more parameters.

4.3 Limitations

The limited computational resources available meant that we could not repeat experiments multiple times to generate confidence intervals, leaving some ambiguity with the results. Additionally, we would have liked to pre-train our model on C4 for quite a bit longer, improving the features learned about language, which may result in an improved performance during fine-tuning.

While the baseline model GPT-2 Medium is similar in structure to ours, it has a larger model dimension, between 65 and 84 million more parameters depending on what version of NeMoT is used, and is pre-trained on different data. Although it is treated as a sort of upper bound in the experiments, it still leaves ambiguity as to what percent neuromodulation contributes to the performance. Therefore, we need a better baseline model, preferably one of similar structure, size and training data, where we can pry out the contribution of the neuromodulatory component and distinguish it from the manually inserted modularity.

In GQA, the out-of-domain dataset's results were quite low with the MQA dataset's results worse than random in all but CQA. Given that the results are poor, especially on the MQA datasets where we know it is worse than random, we have concerns over the reliability of the results. Repeating the experiments multiple times to generate confidence intervals, as previously mentioned, will remedy some of the concerns.

Performance, when compared to the base UnifiedQA model, is very poor, with the difference likely due to the masking of future tokens in all multi-head attention layers. The potential performance of NeMoT has not been brought to light by the decision to make the model uni-directional; pre-training in a masked language modelling objective bi-directionally will overcome this limitation.

We made modifications to two existing reading strategies and found that performance was better with them than without; however, we did not test if the modifications to the reading strategies were better than the original implementations. Therefore, further experiments are needed to determine if the modifications made are a net positive. We have reason to believe that the highlighting reading strategy is better than the original in terms of performance, however, for AOI we expect the improvement to be not in performance but in speed.

4.4 Future work

4.4.1 Experiments

In our experiments we compared NeMoT and the neuromodulatory mechanism's QA and generalisation capabilities to a baseline model of similar structure, but no

neuromodulation; the baseline model consists of between 65 and 84 million more parameters depending on the experiment. We extract positive signals from the experiments that NeMoT achieved a better generalisation, but for a stronger conclusion in regards to the contribution of neuromodulation, further experiments will need to be investigated. Additionally, in our reading strategy experiments, we do not compare our modified implementations to the original. Executing the following will help remedy the previously mentioned concerns:

- The repeating of experiments to generate confidence intervals so more definitive conclusions can be made.
- Pre-training the model with a masked language modelling objective bi-directionally.
- A more diverse set of datasets throughout, including the in-domain and out-of-domain datasets in GQA.
- A better baseline model of similar size that can isolate the effects of neuromodulation, especially from that of the manually introduced modularity in the output set.
- The repeating of the reading strategy experiments with the original implementations of the reading strategies.
- More MQA datasets to test the performance of the two reading strategies.

4.4.2 Neuromodulation

We extended the Transformer via the entwinement of neuromodulation, resulting in a new Transformer architecture: the Neuromodulated Transformer (NeMoT). NeMoT is quite simple in that the gating mechanism only gates the forward traversal of the network at a single point and that it is applied to the output of the vanilla set, not the attention mechanism, for example; it is crude in that the hyperparameters are unrefined. Therefore, potential avenues for future work include:

- Better optimisation of the hyperparameters. For example, experimenting with the number of layers needed for the neuromodulatory set, output set, and vanilla set. We may find that the neuromodulatory set does not need many layers and we can save parameters here. Additionally, given the output set takes as input the gated input of the vanilla set — where neuromodulation occurs and allows for the regulation of a population of neurons — increasing the number of layers of each Transformer block in the output set may further enlarge the impact of the neuromodulatory mechanism, allowing it to regulate more neurons/activations.
- Applying neuromodulated gating to the attention calculation or other locations in the network; make these locations in the network dependent on the context provided to the neuromodulatory set.
- Instead of applying neuromodulated gating to a single location, apply it to multiple locations in the network; a modification to the neuromodulatory set is likely in order to facilitate such.

- A merger of the neuromodulation literature in lifelong learning, where it is applied to the learning rates of connections between neurons, with the Transformer architecture. I.e., the Transformer can continuously learn and adapt to new questions via the modification of learning rates of connections via neuromodulation (this involves an expansion of our definition of GQA).

4.4.3 Reading strategies

In this thesis, we experimented with the utilisation of two reading strategies and evaluate their performance on MQA datasets. Future work includes incorporating more reading strategies such as summarization and paraphrasing among others. A potential summarisation reading strategy in NeMoT involves producing a summary of a passage, potentially conditioned on an associated question, where the produced summary allows the model to more easily extract the answer to a question; the quality of the summary is important for good performance. A potential paraphrasing reading strategy in NeMoT involves the paraphrasing of a question, converting it to a format that portrays the same information, but is easier for the model to understand based on its understanding of text obtained from pre-training; the quality of the paraphrased question is essential for good performance.

Additionally, we only report results for models consisting of only one reading strategy. Integrating multiple reading strategies in a single model, if implemented correctly, will result in better performance as shown in [86, 109]. The best way to achieve the incorporation of multiple reading strategies into a single model is through metacognition, i.e., where the model can choose when and which subset of reading strategies to apply at any given time.

Bibliography

[1] L F Abbott and Wade G Regehr. "Synaptic computation". en. In: *Nature* 431.7010 (Oct. 2004), pp. 796–803.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer Normalization". In: (July 2016). arXiv: 1607.06450 [stat.ML].

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2015).

[4] Linda Baker. "Metacognition". In: *Elsevier* (2010).

[5] Cornelia I Bargmann and Eve Marder. "From the connectome to brain function". en. In: *Nature Methods* 10.6 (June 2013), pp. 483–490.

[6] Shawn Beaulieu et al. "Learning to Continually Learn". In: (Feb. 2020). arXiv: 2002.09571 [cs.LG].

[7] Yonatan Bisk et al. "PIQA: Reasoning about Physical Commonsense in Natural Language". en. In: *AAAI* 34.05 (Apr. 2020), pp. 7432–7439.

[8] Queensland Brain Institute. *What is a neuron?* 2022. URL: https://web.archive.org/web/20220322130954/https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron.

[9] Queensland Brain Institute. *What is synaptic plasticity?* 2022. URL: https://web.archive.org/web/20220321015548/https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/what-synaptic-plasticity.

[10] Tom B Brown et al. "Language Models are Few-Shot Learners". In: (May 2020). arXiv: 2005.14165 [cs.CL].

[11] B D Burrell and C L Sahley. "Learning in simple systems". en. In: *Curr. Opin. Neurobiol.* 11.6 (Dec. 2001), pp. 757–764.

[12] Kunlong Chen et al. "Question Directed Graph Attention Network for Numerical Reasoning over Text". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6759–6768. DOI: 10.18653/v1/2020.emnlp-main.549. URL: https://aclanthology.org/2020.emnlp-main.549.

[13] Jianpeng Cheng, Li Dong, and Mirella Lapata. "Long Short-Term Memory-Networks for Machine Reading". In: (Jan. 2016). arXiv: 1601.06733 [cs.CL].

[14] Chiou-Lan Chern. "Chinese students' word-solving strategies in reading in English". In: *Second language reading and vocabulary learning* (1993), pp. 67–85.

[15] Christopher Clark et al. "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions". In: (May 2019). arXiv: 1905.10044 [cs.CL].

[16] Peter Clark et al. "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge". In: (Mar. 2018). arXiv: 1803.05457 [cs.AI].

[17] Purves D et al. "Neuroscience. 2nd edition. Sunderland (MA): Sinauer Associates; 2001. What Defines a Neurotransmitter?" In: *Beyond neurotransmission: neuromodulation and its importance for information processing (Katz PS, ed)* (2001), pp. 349–381. URL: https://www.ncbi.nlm.nih.gov/books/NBK10957/.

[18] Zihang Dai et al. *Transformer-XL: Attentive Language Models beyond a Fixed-Length Context*. 2019.

[19] Anurag Daram, Angel Yanguas-Gil, and Dhireesha Kudithipudi. "Exploring Neuromodulation for Dynamic Learning". en. In: *Front. Neurosci.* 14 (Sept. 2020), p. 928.

[20] Anurag Reddy Daram, Dhireesha Kudithipudi, and Angel Yanguas-Gil. *Task-Based Neuromodulation Architecture for Lifelong Learning*. 2019.

[21] Pradeep Dasigi et al. "Quoref: A Reading Comprehension Dataset with Questions Requiring Coreferential Reasoning". In: (Aug. 2019). arXiv: 1908.05803 [cs.CL].

[22] M W Decker and J L McGaugh. "The role of interactions between the cholinergic system and other neuromodulatory systems in learning and memory". en. In: *Synapse* 7.2 (Feb. 1991), pp. 151–168.

[23] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (Oct. 2018). arXiv: 1810.04805 [cs.CL].

[24] Kenji Doya. "Metalearning and neuromodulation". en. In: *Neural Networks* 15.4-6 (June 2002), pp. 495–506.

[25] Dheeru Dua et al. "DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs". In: (Mar. 2019). arXiv: 1903.00161 [cs.CL].

[26] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. "Neural modularity helps organisms evolve to learn new skills without forgetting old skills". en. In: *PLoS Comput. Biol.* 11.4 (Apr. 2015), e1004128.

[27] Adam Fisch et al. "MRQA 2019 Shared Task: Evaluating Generalization in Reading Comprehension". In: (Oct. 2019). arXiv: 1910.09753 [cs.CL].

[28] Matt Gardner et al. "Evaluating Models' Local Decision Boundaries via Contrast Sets". In: (Apr. 2020). arXiv: 2004.02709 [cs.CL].

[29] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256.

[30] Chuan Guo et al. "On Calibration of Modern Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1321–1330.

[31] Britta Hahn et al. "Divided versus selective attention: evidence for common processing mechanisms". en. In: *Brain Res.* 1215 (June 2008), pp. 137–146.

[32] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[33] Pengcheng He et al. "DeBERTa: Decoding-enhanced BERT with Disentangled Attention". In: (June 2020). arXiv: 2006.03654 [cs.CL].

[34] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)". In: (June 2016). arXiv: 1606.08415 [cs.LG].

[35] N Jaušovec. "Metacognition". In: *Encyclopedia of Creativity* (2011), pp. 107–112.

[36] Thérèse M Jay. "Dopamine: a potential substrate for synaptic plasticity and memory mechanisms". en. In: *Prog. Neurobiol.* 69.6 (Apr. 2003), pp. 375–390.

[37] Yufan Jiang et al. "Improving Machine Reading Comprehension with Single-choice Decision and Transfer Learning". In: (Nov. 2020). arXiv: 2011.03292 [cs.CL].

[38] John H Kaas. "Neural plasticity". In: (2001).

[39] Paul Katz. *Beyond Neurotransmission: Neuromodulation and its Importance for Information Processing*. en. OUP Oxford, Aug. 1999.

[40] Paul S Katz and Donald H Edwards. "Metamodulation: the control and modulation of neuromodulation". In: *Beyond neurotransmission: neuromodulation and its importance for information processing (Katz PS, ed)* (1999), pp. 349–381.

[41] Daniel Khashabi, Yeganeh Kordi, and Hannaneh Hajishirzi. "UnifiedQA-v2: Stronger Generalization via Broader Cross-Format Training". In: (Feb. 2022). arXiv: 2202.12359 [cs.CL].

[42] Daniel Khashabi et al. "UnifiedQA: Crossing Format Boundaries With a Single QA System". In: (May 2020). arXiv: 2005.00700 [cs.CL].

[43] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014). arXiv: 1412.6980 [cs.LG].

[44] Tomáš Kočisky et al. "The narrativeqa reading comprehension challenge". In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 317–328.

[45] Guokun Lai et al. "RACE: Large-scale ReAding Comprehension Dataset From Examinations". In: (Apr. 2017). arXiv: 1704.04683 [cs.CL].

[46] Weikang Li, Wei Li, and Yunfang Wu. "A unified model for document-based question answering based on human-like reading strategy". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[47] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81.

[48] Kevin Lin et al. "Reasoning Over Paragraph Effects in Situations". In: (Aug. 2019). arXiv: 1908.05852 [cs.CL].

[49] Xiaodong Liu et al. "Multi-Task Deep Neural Networks for Natural Language Understanding". In: (Jan. 2019). arXiv: 1901.11504 [cs.CL].

[50] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: (July 2019). arXiv: 1907.11692 [cs.CL].

[51] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: (Aug. 2016). arXiv: 1608.03983 [cs.LG].

[52] Nicholas Lourie et al. "UNICORN on RAINBOW: A Universal Commonsense Reasoning Model on a New Multitask Benchmark". In: (Mar. 2021). arXiv: 2103.13009 [cs.CL].

[53] Mary Ann Marcinkiewicz. "Building a large annotated corpus of English: The Penn Treebank". In: *Using Large Corpora* (1994), p. 273.

[54] Eve Marder, Timothy O'Leary, and Sonal Shruti. "Neuromodulation of circuits with variable parameters: single neurons and small circuits reveal principles of state-dependent and robust neuromodulation". en. In: *Annu. Rev. Neurosci.* 37 (2014), pp. 329–346.

[55] Eve Marder and Vatsala Thirumalai. "Cellular, synaptic and network effects of neuromodulation". en. In: *Neural Networks* 15.4-6 (June 2002), pp. 479–493.

[56] Bryan McCann et al. "The Natural Language Decathlon: Multitask Learning as Question Answering". In: (June 2018). arXiv: 1806.08730 [cs.CL].

[57] Stephen Merity et al. "Pointer Sentinel Mixture Models". In: (Sept. 2016). arXiv: 1609.07843 [cs.CL].

[58] Thomas Miconi et al. "Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity". In: (Feb. 2020). arXiv: 2002.10585 [cs.NE].

[59] Todor Mihaylov et al. "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering". In: (Sept. 2018). arXiv: 1809.02789 [cs.CL].

[60] Tomas Mikolov et al. "Recurrent neural network based language model". In: *Interspeech*. Vol. 2. 2010, pp. 1045–1048.

[61] Kouider Mokhtari and Ravi Sheorey. "Measuring ESL students' awareness of reading strategies". In: *Journal of developmental education* 25.3 (2002), pp. 2–11.

[62] Denis Paperno et al. "The LAMBADA dataset: Word prediction requiring a broad discourse context". In: (June 2016). arXiv: 1606.06031 [cs.CL].

[63] Scott G Paris and Janis E Jacobs. "The Benefits of Informed Instruction for Children's Reading Awareness and Comprehension Skills". In: *Child Dev.* 55.6 (1984), pp. 2083–2093.

[64] APA Dictionary of Psychology. *Cognition*. 2020. URL: https://web.archive.org/web/20211220115736/https://dictionary.apa.org/cognition.

[65] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[66] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[67] Jack W Rae et al. "Compressive Transformers for Long-Range Sequence Modelling". In: (Nov. 2019). arXiv: 1911.05507 [cs.LG].

[68] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *arXiv preprint arXiv:1910. 10683* (2019).

[69] Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know What You Don't Know: Unanswerable Questions for SQuAD". In: (June 2018). arXiv: 1806.03822 [cs.CL].

[70] Pranav Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: (June 2016). arXiv: 1606.05250 [cs.CL].

[71] Mina Rastegar, Ehsan Mehrabi Kermani, and Massoud Khabir. "The relationship between metacognitive reading strategies use and reading comprehension achievement of EFL learners". en. In: *Open J. Mod. Linguist.* 07.02 (2017), pp. 65–74.

[72] Matthew Richardson, Christopher J C Burges, and Erin Renshaw. "Mctest: A challenge dataset for the open-domain machine comprehension of text". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 193–203.

[73] Keisuke Sakaguchi et al. *WinoGrande: An Adversarial Winograd Schema Challenge at Scale*. 2020.

[74] Maarten Sap et al. *Social IQa: Commonsense Reasoning about Social Interactions*. 2019.

[75] C B Saper. "Brain stem modulation of sensation, movement and consciousness". In: *Principles of Neural Science* (2000), pp. 889–909.

[76] H Şenay Şen. "The Relationship between the Use of Metacognitive Strategies and Reading Comprehension". In: *Procedia - Social and Behavioral Sciences* 1.1 (Jan. 2009), pp. 2301–2305.

[77] Priyanka Sen and Amir Saffari. "What do Models Learn from Question Answering Datasets?" In: (Apr. 2020). arXiv: 2004.03490 [cs.CL].

[78] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: (Aug. 2015). arXiv: 1508.07909 [cs.CL].

[79] R Sheorey and K Mokhtari. "Differences in the metacognitive awareness of reading strategies among native and non-native readers". In: *System* 29.4 (Dec. 2001), pp. 431–449.

[80] Mohammad Shoeybi et al. "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism". In: (Sept. 2019). arXiv: 1909.08053 [cs.CL].

[81] Andrea Soltoggio, Kenneth O Stanley, and Sebastian Risi. "Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks". en. In: *Neural Networks* 108 (Dec. 2018), pp. 48–67.

[82] Andrea Soltoggio et al. "Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios". In: *Proceedings of the 11th international conference on artificial life (Alife XI)*. 2008, pp. 569–576.

[83] Ekta Sood et al. "Improving Natural Language Processing Tasks with Human Gaze-Guided Neural Attention". In: (Oct. 2020). arXiv: 2010.07891 [cs.CL].

[84] Robyn Speer, Joshua Chin, and Catherine Havasi. "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge". en. In: *Thirty-First AAAI Conference on Artificial Intelligence*. Feb. 2017.

[85] Robert Stufflebeam. *Neurons, Synapses, Action Potentials, and Neurotransmission*. 2022. URL: https://web.archive.org/web/20220121043640/https://mind.ilstu.edu/curriculum/neurons_intro/neurons_intro.html.

[86] Kai Sun et al. "Improving Machine Reading Comprehension with General Reading Strategies". In: (Oct. 2018). arXiv: 1810.13441 [cs.CL].

[87] Alon Talmor and Jonathan Berant. "MultiQA: An Empirical Investigation of Generalization and Transfer in Reading Comprehension". In: (May 2019). arXiv: 1905.13453 [cs.CL].

[88] Alon Talmor et al. "CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge". In: (Nov. 2018). arXiv: 1811.00937 [cs.CL].

[89] Adam Trischler et al. "A Parallel-Hierarchical Model for Machine Comprehension on Sparse Data". In: (Mar. 2016). arXiv: 1603.08884 [cs.CL].

[90] Ashish Vaswani et al. "Attention Is All You Need". In: (June 2017). arXiv: 1706.03762 [cs.CL].

[91] Nicolas Vecoven et al. "Introducing neuromodulation in deep neural networks to learn adaptive behaviours". en. In: *PLoS One* 15.1 (Jan. 2020), e0227922.

[92] Roby Velez and Jeff Clune. "Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks". en. In: *PLoS One* 12.11 (Nov. 2017), e0187736.

[93] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: (Apr. 2018). arXiv: 1804.07461 [cs.CL].

[94] Alex Wang et al. "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems". In: (May 2019). arXiv: 1905.00537 [cs.CL].

[95] Kuan Wang et al. "GNN is a Counter? Revisiting GNN for Question Answering". In: (Oct. 2021). arXiv: 2110.03192 [cs.AI].

[96] Psychology Wiki. *Cognitive Processes*. 2021. URL: https://web.archive.org/web/20220317194759/https://psychology.fandom.com/wiki/Cognitive_processes.

[97] Thomas Wood. *F-score*. 2022. URL: https://web.archive.org/web/20220320021023/https://deepai.org/machine-learning-glossary-and-terms/f-score.

[98] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: (Sept. 2016). arXiv: 1609.08144 [cs.CL].

[99] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 2048–2057.

[100] Yichong Xu et al. "Fusing Context Into Knowledge Graph for Commonsense Question Answering". In: (Dec. 2020). arXiv: 2012.04808 [cs.CL].

[101] Min Yang et al. "Exploring Human-Like Reading Strategy for Abstractive Text Summarization". en. In: *AAAI* 33.01 (July 2019), pp. 7362–7369.

[102] Peng-Jian Yang et al. *NT5?! Training T5 to Perform Numerical Reasoning*. 2021. DOI: 10.48550/ARXIV.2104.07307. URL: https://arxiv.org/abs/2104.07307.

[103] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: (June 2019). arXiv: 1906.08237 [cs.CL].

[104] Deming Ye et al. "Coreferential Reasoning Learning for Language Representation". In: (Apr. 2020). arXiv: 2004.06870 [cs.CL].

[105] Dani Yogatama et al. "Learning and Evaluating General Linguistic Intelligence". In: (Jan. 2019). arXiv: 1901.11373 [cs.LG].

[106] İlknur Yüksel and İsmail Yüksel. "Metacognitive Awareness of Academic Reading Strategies". In: *Procedia - Social and Behavioral Sciences* 31 (Jan. 2012), pp. 894–898.

[107] Friedemann Zenke and Wulfram Gerstner. "Hebbian plasticity requires compensatory processes on multiple timescales". en. In: *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 372.1715 (Mar. 2017).

[108] Lian Zhang and Sirinthorn Seepho. "Metacognitive strategy use and academic reading achievement: insights from a Chinese context". In: *Electronic Journal of Foreign Language Teaching* 10.1 (2013).

[109] Shuailiang Zhang et al. "DCMN+: Dual Co-Matching Network for Multi-Choice Reading Comprehension". en. In: *AAAI* 34.05 (Apr. 2020), pp. 9563–9570.

[110] Zhuosheng Zhang, Junjie Yang, and Hai Zhao. "Retrospective Reader for Machine Reading Comprehension". In: (Jan. 2020). arXiv: 2001.09694 [cs.CL].

[111] Barret Zoph et al. "Designing Effective Sparse Expert Models". In: (Feb. 2022). arXiv: 2202.08906 [cs.CL].