Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Author: Alberto Berni
Contact: ab3u21@soton.ac.uk
Date: May 15, 2025

Understanding Attention Mechanisms in Shallow Transformer Networks

Project Supervisor: Prof. Shoaib Jameel
Second Examiner: Dr. Cristoph Tremmel

A project report submitted for the award of:
MEng Computer Science

Abstract

This research project addresses a knowledge gap in Natural Language Processing: understanding how Attention mechanisms encode linguistic features in Shallow Transformer Networks (STNs). The project develops a Transformer Ablation Experiment on Tensor Processing Units (TAETPU) framework for investigating Attention's contribution through controlled experiments.

The framework's technical infrastructure combines a TPU-based runtime environment with Docker containerization and TPU-optimised data processing. It implements four model architectures with varying attention mechanisms, demonstrated through prototype implementations of STNs, with complete training, evaluation, and preliminary probing analysis.

While the project developed functional prototypes that suggest attention mechanisms enhance syntactic encoding capabilities, it fell short of executing full-scale experiments. Time misallocation toward infrastructure, scope underestimation, and inadequate milestone tracking limited the transition from prototypes to comprehensive evaluation. Nevertheless, the framework's methodology, infrastructure, and preliminary results provide valuable contributions to future investigations of Attention mechanisms in STNs.

Statement of Originality

I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell
aiofficer@ecs.soton.ac.uk

Acknowledgements

This research project would not have been possible without the unwavering support and intellectual guidance of Dr. Shoaib Jameel, whose commitment extended beyond conventional supervisory duties. Throughout our numerous meetings, Dr. Jameel consistently provided incisive technical feedback while encouraging theoretical exploration beyond established paradigms. His willingness to engage with the conceptual foundations of this work, particularly when the research questions remained nebulous, proved invaluable. Most significantly, his belief in the project's academic merit persisted through all obstacles, offering exceptional guidance, methodological redirection and contextual perspective that ultimately shaped the trajectory of this investigation.

I extend my profound gratitude to the School of Electronics and Computer Science faculty and the Student Hub team for their institutional support during a particularly challenging academic period. Their administrative flexibility and understanding regarding my health complications enabled a genuine opportunity for academic recovery. This period of academic adjustment proved instrumental in allowing me to develop the foundations documented in this thesis.

To my family and my girlfriend . . .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Space

Inception The ide for this research project started nearly three years ago, when I was studying Theory of Computation. At the time, early-state Generative Pre-Trained (GPT) models like `GPT-Davinci-3` were entertaining the Internet with witty answers and impressive (though now outdated) linguistic capabilities[10]. While exploring the conceptual bases of Turing's *Halting Theorem*, and most importantly *Gödel's Incompleteness Theorem*, I started appreciating the theoretical limitations they imposed on computation itself: based on Formal Logic Systems, any program or algorithm is constrained to *incomplete*[1] axiomatic systems, which lead to inherently undecidable propositions, or *unknowable truths*[2] [13, 11].

Despite these limitations, the power of Context-Free Grammars (CFGs) and the historical developments in Natural Language Processing (NLP) have demonstrated substantial possibilities in Computational Linguistics [14]. Over the past six years, academic publications on large language models (LLMs), have increased by a factor of 570.23 [14]. This surge in (not just) academic Attention prompts further investigations into the deeper functioning and limitations of LLM architectures.

Paying Attention to Transformers The focus of this project revolves around understanding deeper limitations of a particular component of modern LLMs: the Transformer Neural Network (NN) architecture, and its trademark *Attention Mechanism* [5]. Since this introduction, a stream of revolutionising publications flooded academic research [14], starting from the now-ancient BERT (Bi-Directionally Encoded Representations of Trans-

[1]Formally, system S is incomplete if and only if there exists a well-formed formula ϕ in the language of S such that neither ϕ nor its negation $\neg\phi$ is derivable from the axioms and rules of inference of S [11][12].

[2]A positive view of this phenomenon was offered to me by Prof. Corina Cristea, noting that it empowers humans to a realm of *truths* machines are simply incapable of computing.

formers) architecture [15] and culminating in a plethora of new approaches, such as the *Coconut* (Chain Of Continuous Thought) [16] paradigm. Despite this, a significant knowledge gap persists in understanding why and how Attention-based architectures achieve such exceptional performance [9, 17].

Academic Relevance Knowledge gaps persist in Transformer architecture functioning. Research shows Transformer representations are hierarchically structured [18, 6], with Attention layers focusing on distinct linguistic properties [19]. However, the precise mechanisms enabling Attention to encode linguistic relationships in Representation Spaces (RSs)[3] remain unclear [20, 9]. Particularly understudied are Shallow Transformer Networks (STNs) with fewer than 12 encoder-decoder stacks, despite their computational efficiency and practical importance [7, 21]. This knowledge gap becomes increasingly relevant as resource-constrained applications drive demand for efficiency-focused architectures [22, 23].

Technical Relevance This project offers three interconnected contributions: **(1)** the TAETPU framework—a dedicated TPU-based runtime environment using Google Cloud's TPU VMs [24] and Docker, enabling reproducible Transformer experimentation [25]; **(2)** container utility scripts that streamline directory management between local and container environments, eliminating image rebuilding for development iterations; and **(3)** a Python data processing pipeline with task-specific generators and TPU optimizations [26]. These implementations connect theoretical Transformer research with hardware constraints, facilitating investigations into Attention mechanisms [27], information flow [8], and model interpretability [28].

1.2 Project Goals

Research Question This project investigates the fundamental research question: *Why are Transformers so effective at encoding linguistic features?* This question directly addresses the theoretical knowledge gap identified in current literature regarding information flow in Shallow Transformer Networks (STNs), which have less than 12 Encoder-Decoder layers[4]. [17, 9]. This research focuses specifically on STNs because the Attention mechanisms are more tractable to analyse than deep pre-trained models like BERT or GPT [15, 14].

Theoretical Framework Conceptually, the project aims to propose a novel information-theoretic framework for analysing Attention mechanisms in shallow Transformers. This

[3] See section 2.3.1

[4] See section 2.4.3 for context.

framework quantifies how linguistic features (syntactic, semantic, and contextual) are encoded, transformed, and preserved through Attention layers. Our approach combines:

1. Structural Probing techniques [6].
2. Comparative analysis between Transformer and Static[5] architectures.
3. Visualisation techniques for Attention weight distributions across layers and heads[29]
4. Information flow mapping to trace feature preservation through network layers [7]

The proposed framework aims to trace information flow from input tokens through successive transformations, providing insights into how Attention mechanisms encode linguistic structures that static embeddings cannot capture.

Methodological Design This project proposes a systematic comparison between STNs and Static NN embedding architectures, following the brief's requirements:

1. Two main model categories are compared:
   - **Set A**: Feed-Forward Networks with static word embeddings (GloVe [30] and Word2Vec [31])
   - **Set B**: Transformer variants with varying Attention mechanisms, where M2AN denotes models with N encoder-decoder Attention heads ($N \in \{0, 1, 2, ..., 12\}$) [5]
2. Models are evaluated across diverse NLP tasks testing syntactic understanding (POS tagging), semantic processing (NER, Sentiment), and contextual capabilities (MLM, NSP)
3. Two datasets provide both short-context (`dar-ai/emotion`) and long-range dependencies (`nbeerbower/gutenberg2-dpo`)

This ablation approach allows quantification of information preservation across architectural configurations, providing evidence for Attention's advantages in contextual language representation [32, 33].

Scope and Limitations The project focuses on understanding Attention mechanisms in Transformer models with the standard 12 Encoder-Decoder stacks. The research prioritises theoretical and methodological understanding over downstream task performance optimisation, examining fundamental linguistic capabilities rather than application-specific metrics. These limitations are intentionally chosen to enable deeper theoretical analyses. The controlled experimental design provides a foundation for future extensions to more complex architectures while establishing fundamental insights into Attention mechanisms' performance.

[5]See sections 2.3.1, and 2.4.3 for context.

Chapter 2

NLP Background

2.1 Introduction

Definitions Natural Language Processing (NLP) can be defined as *the Scientific study of language from a computational perspective* [34]. A more natural definition is '*to consider what it would take to create an intelligent agent like HAL from 2001*[35]'. The research field has evolved rapidly over the past decades, transitioning from early rule-based approaches using Context-Free Grammars (CFGs) and Abstract Syntax Trees (ASTs) [36, 37, 38, 39], to statistical methods leveraging probability distributions[40], to the current Transformer-based and multimodal architectures[35]. This chapter traces this evolution, focusing on the findings that led to modern Transformer-based architectures.

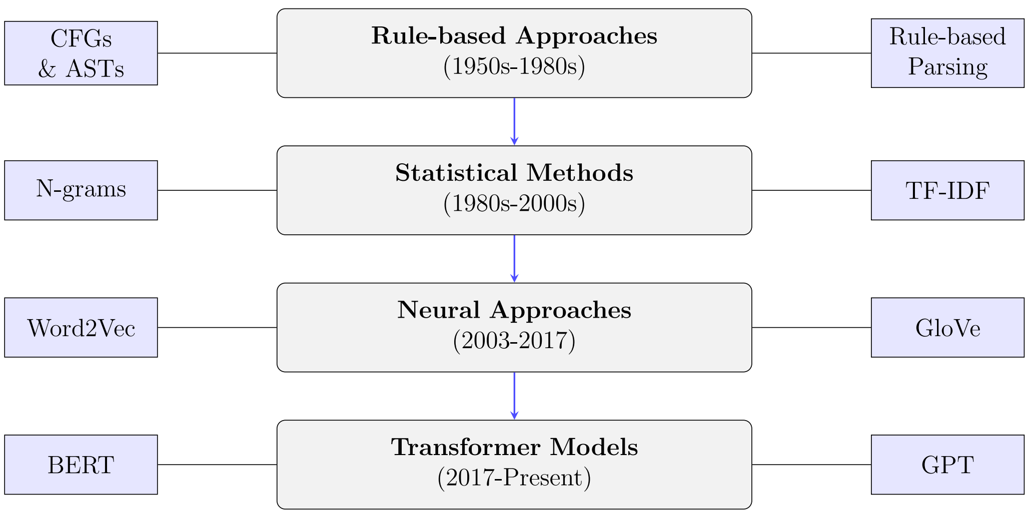


Figure 2.1: Evolution of NLP approaches and techniques, highlighting the progression from rule-based to Transformer-based paradigms[1, 2].

Foundational Works Early approaches to NLP emerged from formal language theory, particularly through Chomsky's Context-Free Grammars (CFGs) [36] and their computational implementations. The now-archaic systems relied on formal rules to numerically encode linguistic structures, using hierarchical trees to represent syntactic relationships. Abstract Syntax Trees (ASTs), originating from compiler design, became central to rule-based parsing, allowing machines to decompose sentences into meaningful nested constituents[35]. While this rule-based approach is direct and intuitive, it is limited in generalising linguistic features beyond explicit patterns, and thus ignores the inherent complexity of natural language [12, 11].

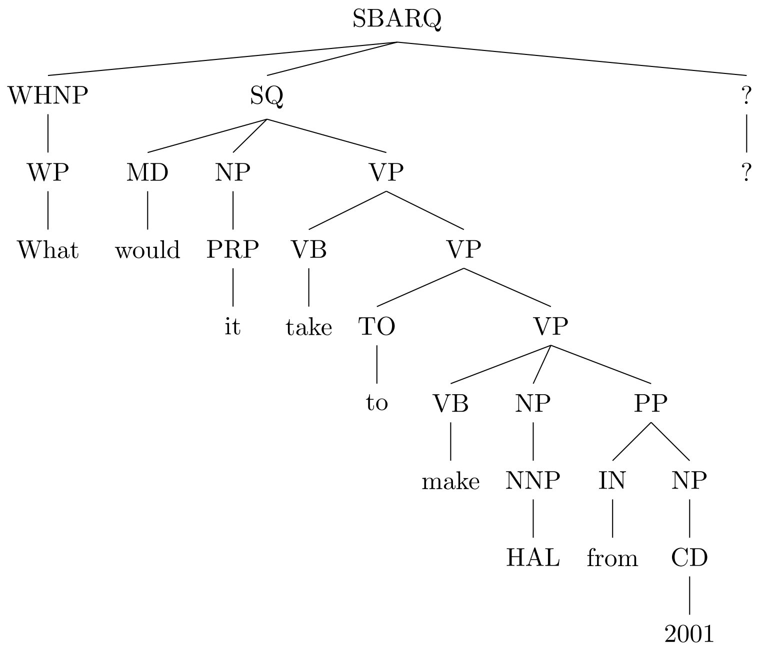


Figure 2.2: Abstract Syntax Tree (AST) of the sentence *what would it take to make HAL from 2001?*

The Perceptron A notable implementation of theoretical works is Rosenblatt's Perceptron model. Based on McCulloch and Pitts's neuron model, Perceptrons perform binary classification by applying a *linear activation function* $f(a)$ to a non-linear transformation ϕ to the input vector $\mathbf{x}$ [3, 4]. The process can be mathematically summarized as follows:

$$y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x})), \quad \text{where} \quad f(a) = \begin{cases} +1, & a \geq 0, \\ -1, & a < 0. \end{cases} \tag{2.1}$$

The activation function $f(a)$ encodes two processes: *excitatory* ($a \geq 0$) and *inhibitory* ($a < 0$). This allows us to assign our two classes to these processes: let C_1 correspond to $+1$ and C_2 to -1. Then,

$$\mathbf{x}_i \in C_1 \iff \mathbf{w}^T\phi(\mathbf{x}_i) \geq 0. \tag{2.2}$$

The Perceptron's dominance in the field was quickly stopped by Minsky and Papert [41]; their critique of the model highlighted critical limitations regarding its inability to solve non-linearly separable problems (such as the XOR problem), regardless of the number of neurons implemented. Positing that models required *non-linear* activation functions and more refined training algorithms,Minsky and Papert's research posed problems that would last two decades, a period known as the *AI winter* [35].

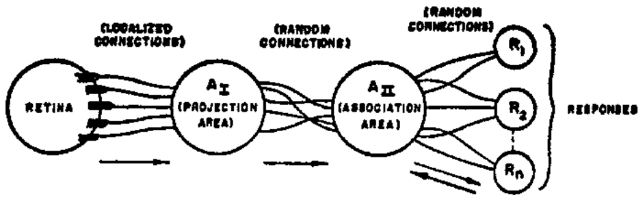


Figure 2.3: Perceptron Model schematics as proposed by Rosenblatt [3]

Neural Networks AI research found its resurgence in the mid 1980's when Rumelhart et al. popularized the Multi-layer Perceptron (MLP) by demonstrating it could classify non-linearly-separable datasets if trained with backpropagation [42]. The MLP extends Rosenblatt's work by linking Perceptron-like nodes through their inputs and outputs to form hidden layers[4]. The MLP consists of an input layer, one or more hidden layers, and an output layer; for a network with D inputs $\{x_i\}_{i=1}^{D}$, a hidden layer of M units, and K outputs, the model implements a linear combination of non-linear activation functions $\phi(\mathbf{x})$:

$$y(\mathbf{w}, \mathbf{x}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right). \tag{2.3}$$

It is to be noted that the MLP model, implements differentiable activation functions such as the sigmoid or tanh function to train the model based on its error[4]. Thus, for an MLP with two hidden layers, the entire network function can be expressed as:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right), \tag{2.4}$$

where $w_{ji}^{(1)}$ are the first-layer weights, $w_{j0}^{(1)}$ are the biases for the j-th hidden unit; the superscript (ℓ) indicates the hidden layer number. Note that that each hidden unit performs a parametrization of the previous unit's output $w_{MD}^{(\ell)} h(\cdot)$. This mechanism allows NNs to adaptively scale activation functions during training, permitting them to classify and process complex, non-linear datasets [4, 42]. Appendix 12 deepens the mathematical discourse on MLP by describing training phases.

With the advent of MLPs, the *connectionist* paradigm gained traction in NLP research: information started to be seen as patterns of activity across layers, or Distributed Repre-

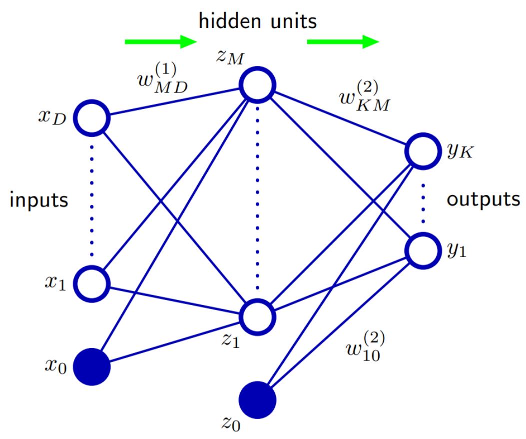


Figure 2.4: Graph of a single-hidden-layer MLP from C. Bishop's book [4].

sentations [42]. This methodological shift laid the foundations for encoding data entities as continuous feature vectors jointly learned by a system, verging closer to the modern definition of WEs. Following the connectionist paradigm, probabilistic NNs started being theorized: early works by MacKay [43] and Neal [44] introduced inference of posterior distributions over model parameters via Markov Chain Monte Carlo (MCMC) techniques, allowing models to express uncertainty when making predictions.

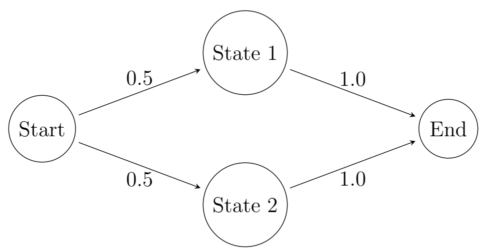


Figure 2.5: Example Markov Chain Monte Carlo for Language Prediction. A more accurate description of the process is detailed in Appendix 12

2.2 Word Embeddings

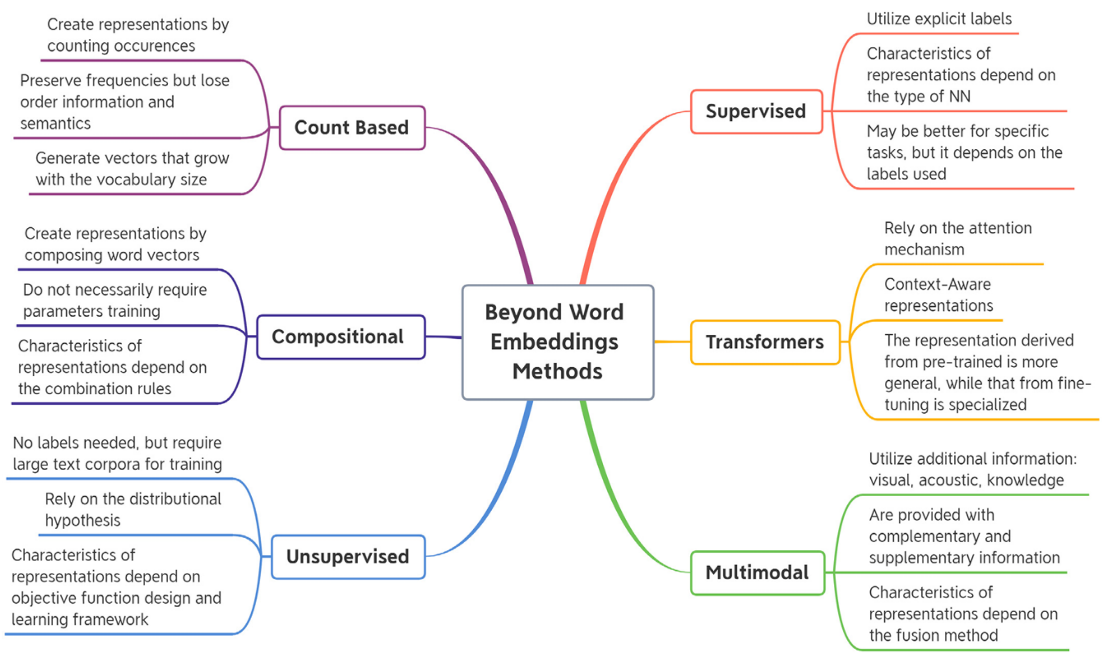


Figure 2.6: Comprehensive Taxonomy of Word Representation Methodologies Beyond Traditional Embeddings [2].

Standard Pipelines At its core, the NLP pipeline can be understood as a sequence of computations to generate Word Embeddings (WEs): numerical representations of linguistic information. Since the early 1980s, data preprocessing steps have remained remarkably consistent [45]: standard preprocessing operations follow a sequential transformation from raw text to structured numerical representations [35, 2]. As depicted in Figure 2.7, this process progresses sequentially from corpus splitting to final vectorisation, with each intermediate stage (noise removal, normalisation, stop word filtering, and tokenisation) systematically constraining the linguistic inputs [35, 2].

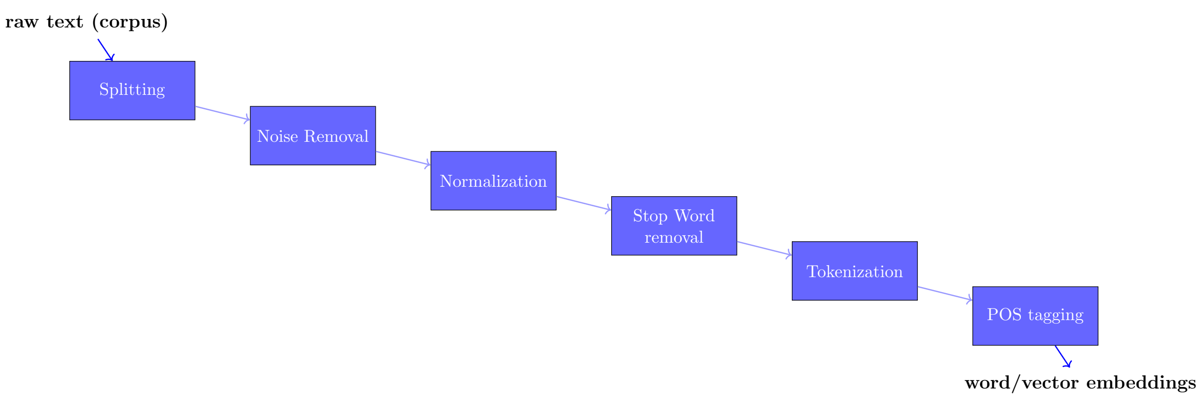


Figure 2.7: A standard NLP Pipeline

2.2.1 Frequency-based Methods

The earliest approaches to representing words numerically relied on simple frequency counts. The foundational *Bag-Of-Words* (BOW) model represented documents as vectors where each dimension corresponded to a word's frequency [35]. This approach, while intuitive, suffered from sparsity issues due to the vast vocabulary size in natural language, creating high-dimensional, sparse vectors that proved computationally challenging. A significant advancement came with the introduction of Term Frequency-Inverse Document Frequency (TF-IDF), a statistical measure that evaluates word importance within a document relative to a corpus [46]. Mathematically, TF-IDF is expressed as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{2.5}$$

where term frequency is calculated as:

$$\text{TF}(t, d) = \frac{\text{count of term } t \text{ in document } d}{\text{total number of terms in document } d} \tag{2.6}$$

and inverse document frequency is:

$$\text{IDF}(t, D) = \log \frac{\text{total number of documents in corpus } D}{\text{number of documents containing term } t} \tag{2.7}$$

While these frequency-based approaches provided foundational capabilities for information retrieval systems, they failed to capture semantic relationships between words, treating each term as an independent entity without contextual connections [2, 1].

2.2.2 Neural Word Embeddings

Semantic Spaces The breakthrough in word representations came with neural embedding methods, which learn dense, low-dimensional vectors through prediction tasks. The Continuous Bag-of-Words (CBOW) architecture [31] predicts a target word from its context words, while the Skip-gram model reverses this relationship, predicting context from a target word. For CBOW, the objective function can be formulated as:

$$\mathcal{L}_{\text{CBOW}} = \frac{1}{T} \sum_{t=1}^{T} \log p(w_t | w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) \tag{2.8}$$

Where T is the total number of training words and c is the size of the context window. The probability is typically computed using the softmax function:

$$p(w_t|\text{context}) = \frac{\exp(u_{w_t}^\top \bar{v}_{\text{context}})}{\sum_{w \in V} \exp(u_w^\top \bar{v}_{\text{context}})} \tag{2.9}$$

Here, u_w is the output vector for word w, and $\bar{v}_{\text{context}}$ is the average of the context word vectors. Neural word embedding models produce dense vector representations that capture semantic and syntactic relationships between words, allowing for algebraic operations on language (e.g., vector(”*king*”) − vector(”*man*”) + vector(”*woman*”) ≈ vector(”*queen*”)) [31]. These models significantly outperform frequency-based methods in various NLP tasks by encoding distributional semantics in a much more compact and generalizable form [2, 1].

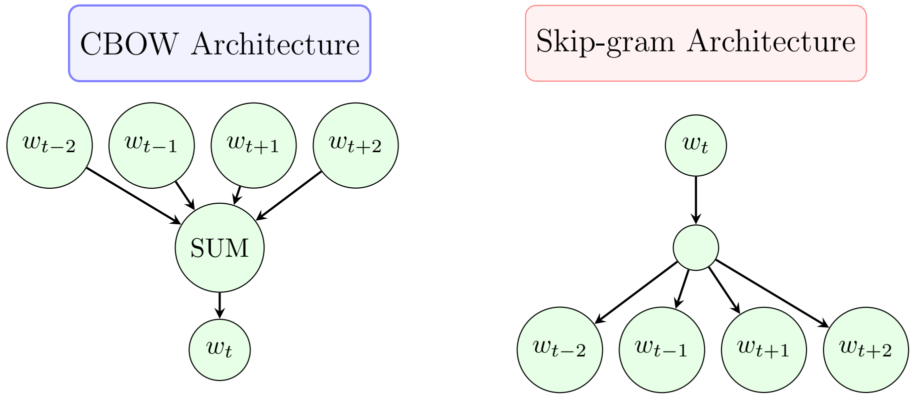


Figure 2.8: Architectural comparison of Continuous Bag-of-Words (CBOW) and Skip-gram models in Word2Vec. CBOW predicts the target word from context words, while Skip-gram predicts context words given a target word.

2.3 Static Embeddings

2.3.1 Distributional Hypothesis

The distributional hypothesis, articulated by J.R. Firth's statement that *you shall know a word by the company it keeps* [47], forms the foundation for modern word embedding techniques. The principle suggests that words appearing in similar contexts likely share semantic properties. Mathematically, we can express this as an expectation over contexts:

$$\mathrm{Sim}(w_i, w_j) \approx \mathrm{Sim}(P(c|w_i), P(c|w_j)) \tag{2.10}$$

where $P(c|w)$ represents the probability distribution of contexts c given word w. This paradigm promotes higher-dimensional language encoding due to its exponential modelling of probability distributions. The resulting complexity spike constitutes the early development of Representational Spaces (RSs): high-dimensional vector spaces that encode rich, multidimensional word vector. embedding[48].

GloVe Global Vectors for Word Representation (GloVe) [30] approached word embeddings through direct factorization of the word co-occurrence matrix. Given a vocabulary V and a co-occurrence matrix X where each entry X_{ij} represents how often word j appears in the context of word i, GloVe minimizes:

$$J_{\text{GloVe}} = \sum_{i,j=1}^{|V|} f(X_{ij})(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{2.11}$$

where w_i and $\tilde{w}_j$ are word and context vectors respectively, b_i and $\tilde{b}_j$ are bias terms, and $f(X_{ij})$ is a weighting function:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{2.12}$$

GloVe's strength lies in its ability to directly encode global statistical information about word co-occurrences, capturing both global corpus statistics and local contextual information.

Word2Vec The Skip-gram architecture of Word2Vec [31, 49] inverted the prediction problem by learning to predict context words given a target word. Its objective function is:

$$J_{\text{Skip-gram}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j}|w_t) \tag{2.13}$$

where c is the context window size. This probability is computed using the softmax function:

$$P(w_{t+j}|w_t) = \frac{\exp(v_{w_{t+j}}^\top v'_{w_t})}{\sum_{w=1}^{|V|} \exp(v_w^\top v'_{w_t})} \tag{2.14}$$

where v_w and v'_w are the input and output vector representations of word w. To address computational challenges with large vocabularies, negative sampling was introduced:

$$J_{\text{NS}} = \log \sigma(v_{w_O}^\top v'_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-v_{w_i}^\top v'_{w_I})] \tag{2.15}$$

where σ is the sigmoid function, w_I is the input word, w_O is the output word, and $P_n(w)$ is the noise distribution for negative sampling. Despite their effectiveness, both GloVe and Word2Vec share a critical limitation: they generate a single static vector per word, failing to account for polysemy and context-dependent meaning variations. This limitation motivated the development of richer embedding models based on the Transformer architecture [2, 1].

2.4 Transformer Embeddings

2.4.1 Transformer Architectures

The Transformer architecture [5] revolutionized NLP by replacing recurrent connections with multi-headed self-Attention mechanisms. This architectural innovation facilitated parallel computation and enabled more effective modeling of long-range dependencies in text. A Transformer consists of encoder and decoder stacks, each containing six identical layers. Each encoder layer has two sub-layers: a multi-head self-Attention mechanism and a position-wise fully connected feed-forward network. The decoder additionally inserts a third sub-layer for Attention over the encoder's output. Each sub-layer employs a residual connection and layer normalization:

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x)) \tag{2.16}$$

To incorporate positional information in the absence of recurrence, Transformers add positional encodings to the input embeddings:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right) \tag{2.17}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right) \tag{2.18}$$

where pos is the position, i is the dimension, and d_{model} is the embedding dimension.

2.4.2 The Attention Mechanism

The core innovation of Transformers is the scaled dot-product Attention mechanism, which can be expressed as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.19}$$

where Q, K, and V represent query, key, and value matrices, and d_k is the dimension of the key vectors. This formulation computes a weighted sum of values, where the weight assigned to each value is determined by the compatibility of the query with the corresponding key. Transformers employ multi-head Attention, allowing the model to jointly attend to information from different representation subspaces:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O \tag{2.20}$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.21}$$

This mechanism enables the model to capture different aspects of similarity between sequence elements, from syntactic relationships to semantic associations, across different Attention heads [17].

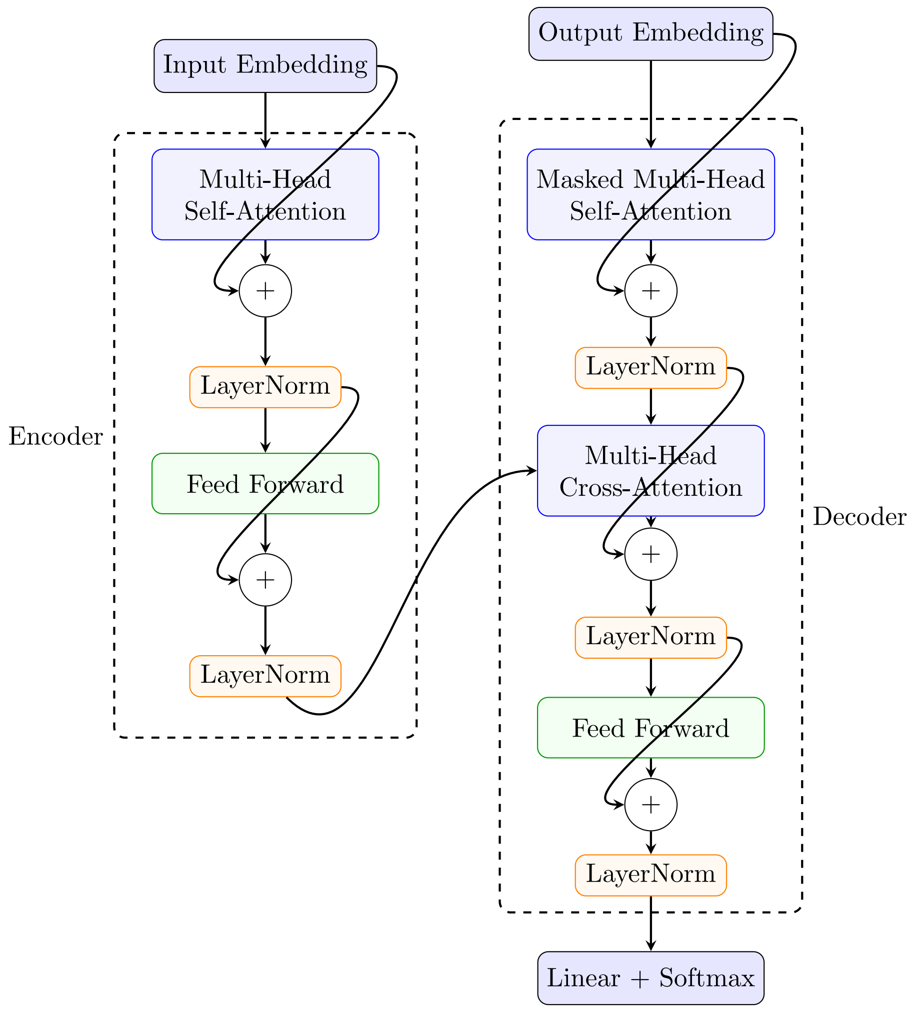


Figure 2.9: The Transformer architecture featuring encoder and decoder stacks, with multi-head Attention mechanisms and feed-forward networks. Residual connections and layer normalization are applied to each sub-layer [5].

The residual connections in Transformers introduce additional complexity through the following formulation [5]:

$$\mathbf{h}_i^{l+1} = \mathbf{h}_i^l + \text{Attn}(\text{LN}(\mathbf{h}_i^l)) + \text{MLP}(\text{LN}(\mathbf{h}_i^l + \text{Attn}(\text{LN}(\mathbf{h}_i^l)))) \tag{2.22}$$

where $\mathbf{h}_i^l$ represents the hidden state for token i at layer l, LN is layer normalization, Attn is the Attention mechanism, and MLP is the feed-forward neural network. This architecture creates complex interactions that make it challenging to isolate the contribution of individual components. These challenges are particularly relevant for shallow networks, where the interplay between fewer Attention heads and fewer Transformer layers might create different dynamics than in deeper models. The reduced depth might make Attention patterns more directly influential on outputs, potentially increasing their interpretability but also requiring careful analysis to avoid overstating their explanatory power.

2.4.3 BERT-families

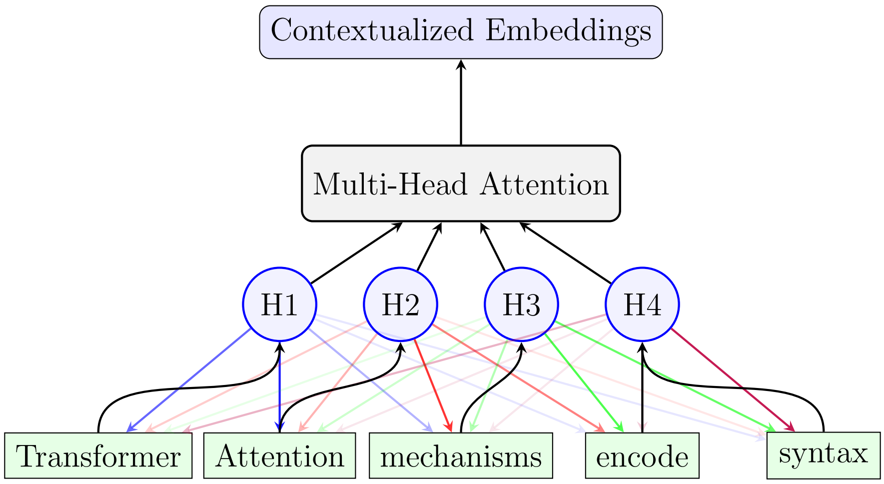


Figure 2.10: Multi-head Attention mechanism in Transformers. Different Attention heads (H1-H4) focus on different linguistic properties and relationships between tokens, capturing various aspects of the input sequence simultaneously.

Architectural Formalism Bidirectional Encoder Representations from Transformers (BERT) [15] constitutes a transformative paradigm in NLP architectures by formalizing a deeply bidirectional representation mechanism trained on unlabeled text corpora. The fundamental innovation of BERT lies in its pre-training methodology incorporating two distinct objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

The BERT architecture can be formally defined as a multi-layer bidirectional Transformer encoder with parameters $\theta = \{L, H, A\}$, where L represents the number of encoder layers, H denotes the hidden dimensionality, and A indicates the number of self-Attention heads. Standard configurations include BERT$_{\text{BASE}}$ $(L = 12, H = 768, A = 12)$ with 110M parameters and BERT$_{\text{LARGE}}$ $(L = 24, H = 1024, A = 16)$ with 340M parameters.

Multi-head Attention Mechanism The core component enabling BERT's contextual understanding is the multi-head Attention mechanism. For a given input sequence $X =$

$\{x_1, x_2, ..., x_n\}$, each Attention head computes scaled dot-product Attention according to:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.23}$$

where Q, K, and V represent the query, key, and value matrices derived from linear projections of the input representation, and d_k is the dimensionality of the key vectors. The multi-head Attention operation is then formulated as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.24}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.25}$$

This formalism allows each Attention head to focus on different linguistic aspects of the input sequence, as illustrated in Figure 2.10.

Training Objectives BERT implements two distinct pre-training tasks that distinguish it from previous approaches:

1. **Masked Language Modeling (MLM)**: Formally, given an input sequence $X = \{x_1, x_2, ..., x_n\}$, a subset of tokens $M \subset \{1, 2, ..., n\}$ is selected for masking with $|M|/n = 0.15$. The MLM objective is defined as:

$$\mathcal{L}_{\text{MLM}} = -\mathbb{E}_X \left[\sum_{i \in M} \log P(x_i | X_{\setminus M}; \theta)\right] \tag{2.26}$$

   where $X_{\setminus M}$ represents the sequence with masked tokens. To mitigate pre-train/fine-tune discrepancy, BERT employs a mixed replacement strategy:

   - 80% of tokens in M are replaced with [MASK]
   - 10% are replaced with random vocabulary tokens
   - 10% remain unchanged

2. **Next Sentence Prediction (NSP)**: Given sentence pairs (S_A, S_B), the model is trained to predict whether S_B follows S_A in the original text:

$$\mathcal{L}_{\text{NSP}} = -\mathbb{E}_{(S_A, S_B)} \left[\log P(\text{IsNext}(S_A, S_B) | [S_A; S_B]; \theta)\right] \tag{2.27}$$

   where $[S_A; S_B]$ denotes the concatenated representation of the sentence pair.

The joint training objective is formulated as:

$$\mathcal{L} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}} \tag{2.28}$$

Information Flow in BERT The residual connections and self-Attention mechanism in BERT enable complex information propagation through the network, which can be formalized as:

$$\mathbf{h}_i^{l+1} = \mathbf{h}_i^l + \mathrm{MLP}(\mathrm{LN}(\mathbf{h}_i^l + \mathrm{Attn}(\mathrm{LN}(\mathbf{h}_i^l)))) \tag{2.29}$$

where $\mathbf{h}_i^l$ represents the hidden state for token i at layer l, LN is layer normalization, Attn is the Attention mechanism, and MLP is the feed-forward neural network. Figure **??** illustrates how Attention weight distributions vary across different heads within the same layer, enabling the capture of diverse linguistic phenomena.

2.5 Computational Facilities

The TPU Tensor Processing Units (TPUs) are application-specific integrated circuits (ASICs) developed by Google specifically to accelerate machine learning workloads, particularly those employing neural network architectures [24]. TPUs are optimized for matrix operations, making them especially well-suited for Transformer-based NLP models that rely heavily on matrix multiplications in their Attention mechanisms.

The TPU architecture centers around a Matrix Multiplication Unit (MXU), capable of performing 16K multiply-accumulate operations per clock cycle. This design enables TPUs to significantly outperform CPUs and GPUs for neural network training and inference. The architecture includes:

1. **Matrix Multiplication Unit**: Optimized for 8-bit or 16-bit matrix operations, the core computational engine.
2. **Vector Processing Unit**: Handles non-matrix computations like activation functions.
3. **High Bandwidth Memory**: Provides fast memory access for model weights and activations.
4. **Interconnects**: Enable scaling across multiple TPU devices for distributed training.

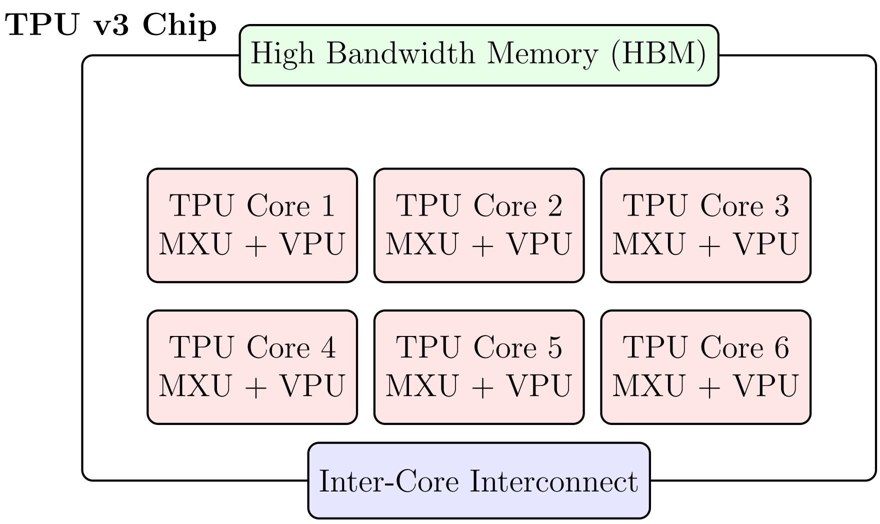


Figure 2.11: Simplified architecture of a TPU v3 chip, showing multiple TPU cores with Matrix Multiplication Units (MXUs) and Vector Processing Units (VPUs), connected to High Bandwidth Memory (HBM).

Chapter 3

Transformer Probing Literature

Introduction Modern probing methods involve training supervised classifiers to identify linguistic properties encoded in neural Representational Spaces (RSs)[19, 18]. This chapter examines probing techniques for understanding the internal representations of transformer models, addressing the central question: *Which techniques allow us to examine Attention layers across shallow and deep networks?*

3.1 Structural Probing

The paradigm Structural probing identifies syntactic structures encoded in transformer representations. Hewitt and Manning [6] test whether transformer representations encode parse trees by finding linear transformations where distances between word vectors approximate syntactic distances in parse trees.

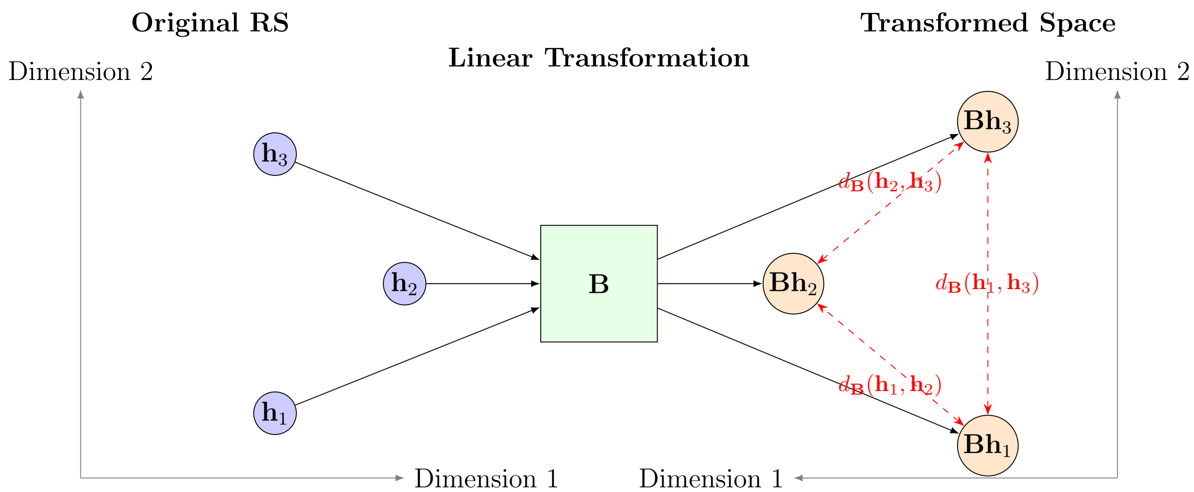


Figure 3.1: Structural Distance Probe flow chart of operations[6].

Distance Probing For word representations $\mathbf{h}_1, \ldots, \mathbf{h}_n$, we search for a transformation matrix $\mathbf{B}$ such that:

$$d_{\mathbf{B}}(\mathbf{h}_i, \mathbf{h}_j)^2 = \|\mathbf{B}(\mathbf{h}_i - \mathbf{h}_j)\|^2 \tag{3.1}$$

approximates the number of edges in the shortest path between words i and j in a parse tree. This approach examines global geometric properties rather than layer-specific Attention patterns. A key finding is that syntax is encoded in a low-dimensional subspace of the representation space [6], suggesting transformers allocate specific capacity to syntactic structure while maintaining separate dimensions for semantic information [50].

Depth Probing Depth probing addresses hierarchical organization in parse trees using a linear transformation such that:

$$\|\mathbf{B}\mathbf{h}_i\|^2 \approx \mathrm{depth}(i) \tag{3.2}$$

This treats tree depth as a norm in the transformed space. Success in recovering tree depths provides evidence that transformer models encode hierarchical relationships geometrically [15, 51] despite never receiving explicit tree-structured supervision.

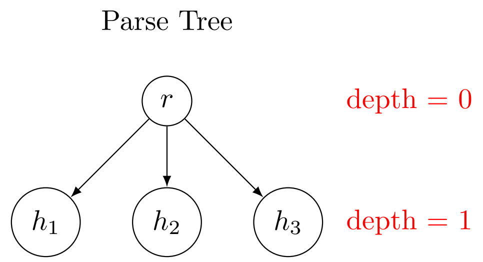


Figure 3.2: Graphical estimation of how Structural Depth Probe perceive ASTs. The r node is the tree root and the h_n nodes are words as encoded in the transformed ES [6].

Limitations While structural probing reveals encodings of syntactic structure, correlation between geometric properties and linguistic structures doesn't imply causality in language processing [52]. Additionally, probe complexity itself can confound results: simple probes may miss encoded information while complex probes might learn tasks independently [53]. For shallow transformers, fewer layers are available for hierarchical information development, suggesting they may encode syntactic information differently.

3.2 Information Probing

Theoretical Framework Information theory provides tools for analyzing representations by measuring mutual information between representations and linguistic features [52]. Key concepts include:

Entropy ($H(X)$), which measures uncertainty in a random variable $x \in X$:

$$H(X) = -\sum_{x} p(x) \log p(x) \tag{3.3}$$

Lower entropy indicates more focused Attention, while higher entropy suggests more dispersed Attention. Cheng et al. [54] refined this approach to express Attention Entropy A_E, the Attention-head weight distribution, and a_i, the individual Attention weights:

$$A_E = -\sum_{j=1}^{n} a_{ij} \log a_{ij} \tag{3.4}$$

where a_{ij} represents the Attention weight from token i to token j. This metric has proven valuable for analysing shallow transformers, where Attention patterns tend to be more interpretable due to fewer layer interactions [**?**].

Mutual Information (MI) ($I(X;Y)$) quantifies uncertainty in information between variables$x \in X$, $y \in Y$:

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{3.5}$$

Jensen-Shannon divergence (JSD) has emerged as a popular metric for comparing Attention patterns [53]:

$$\mathrm{JSD}(P\|Q) = \frac{1}{2} \cdot D(P\|M) + \frac{1}{2} \cdot D(Q\|M) \tag{3.6}$$

where D is the Kullback-Leibler divergence and $M = \frac{1}{2}(P+Q)$. JSD is particularly useful for quantifying differences between Attention distributions across heads, layers, or models [52]. Hoyos-Osorio et al. [55] proposed the Representation Jensen-Shannon Divergence (RJSD), which embeds data into a reproducing kernel Hilbert space and computes JSD between covariance operators. This approach offers a more geometrically intuitive way to compare Attention distributions in shallow transformers, where head specialization may be more pronounced than in deeper models.

3.2.1 Core Methodologies

Information Bottleneck Formulation Pimentel et al. [53] reframed probing within the Information Bottleneck (IB) framework. They propose an optimization of RS quality through information preservation-compression and trade-offs:

$$\mathcal{L}_{\text{IB}} = I(X;Z) - \beta I(Z;Y) \tag{3.7}$$

where X represents input, Z represents intermediate representation, Y represents linguistic property, and β controls the trade-off between compression and information preservation. This perspective illustrates how Transformers implicitly discard irrelevant information while preserving task-necessary features.

Minimum Description Length Probing Voita and Titov [52] framed probing via the Minimum Description Length (MDL), measuring information intractability, and accessibility from representations. To minimise the encoding cost and information loss, they propose a multi-dimensional assessment of ESs by quantifying data volume needed for probe training and probe complexity:

$$L(D, \theta) = L(\theta) + L(D|\theta) \tag{3.8}$$

where $L(\theta)$ represents parameter description cost and $L(D|\theta)$ represents data description cost given parameters. For online code transmission, the MDL-based complexity metric becomes:

$$\text{complexity} = -\sum_{i=1}^{|D_{\text{train}}|} \log_2 p(y_i|x_i, D_{\text{train}_{<i}}) \tag{3.9}$$

where $p(y_i|x_i, D_{\text{train}_{<i}})$ represents predictive probability of label y_i after observing previous examples.

V-Information and MI Estimation Xu et al. [54] introduced V-information as a practical way to estimate information flow in transformers. This approach addresses the challenge of estimating MI in high-dimensional spaces, which is notoriously tricky due to the curse of dimensionality [2]. To this end they propose the following techniques:

Noise-contrastive estimation: Uses contrastive samples to approximate density ratios

$$\hat{I}_{\text{NCE}}(X;Y) = \mathbb{E}_{p(x,y)}\left[\log \frac{f_\theta(x,y)}{\mathbb{E}_{p(y')}[f_\theta(x,y')]}\right] \tag{3.10}$$

Variational bounds: Approximates MI through neural estimation

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}[\log q_\theta(y|x)] + H(Y) \tag{3.11}$$

k-nearest neighbor methods: Non-parametric estimation using local density approximation

$$\hat{I}_k(X;Y) \approx \psi(N) + \psi(k) - \frac{1}{N}\sum_{i=1}^{N}[\psi(n_x(i)) + \psi(n_y(i))] \tag{3.12}$$

where ψ is the digamma function and $n_x(i)$, $n_y(i)$ count neighbors in respective spaces.

Relevance to STNs Information-theoretic approaches have been invaluable for understanding Attention mechanisms in transformers [21, 2, 7, 19]. Academic research suggests STNs may show less-pronounced information compression, using different mechanisms for separating relevant and irrelevant information. This underscores a knowledge gap regarding information flow mapping of shallow Attention mechanisms.

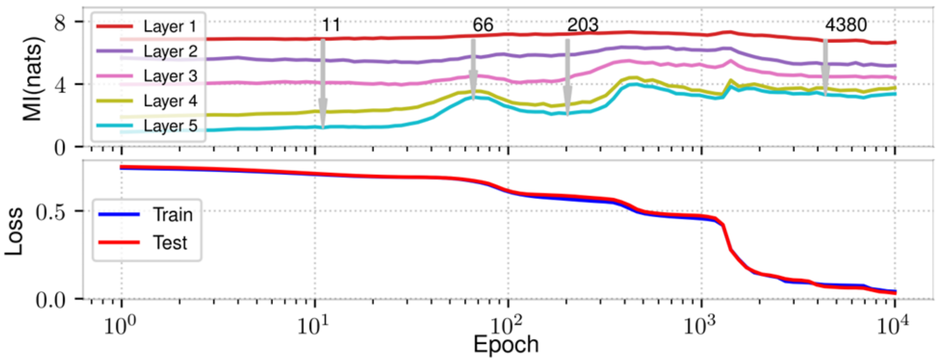


Figure 3.3: MI line chart of a CNN across epochs as described by Goldfeld et al. As task performance improves, information loss decreases, indicating a progressive information encoding[7].

3.3 Attention Head Probing

3.3.1 Functional Specialization

Attention encodes higherarchies Research has demonstrated that Transformer Attention heads *focus* on specific linguistic functions during training. Voita et al. [33] discovered that different heads in machine translation models specialise in tasks such as attending to proper nouns, rare words, or syntactically related words. Clark et al. [17] identified specific heads in BERT that appear to track syntactic dependencies, identifying common specialisations as:

Positional Attention: Attention to adjacent tokens ($i \to i \pm 1$)
Special token Attention: Attention to/from [CLS]/[SEP] tokens
Delimiter Attention: Attention to punctuation and separators
Syntactic Attention: Attention following syntactic dependencies

This distinction helps reveal how different Attention heads specialise in capturing various aspects of linguistic structure.

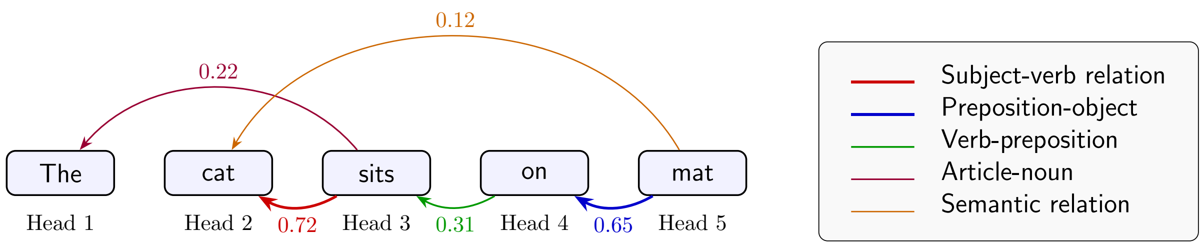


Figure 3.4: Multi-head Attention specialisation is often analysed through weight patterns. This graph conceptually visualises the Transformer-specific Attention head specialisation.

3.3.2 Estimating Functional Specialisation

Direct Estimation Michel et al. [56] suggest an early contribution to Attention head probing by calculating the expected sensitivity of the model to pruning specific heads:

$$I_h = \mathbb{E}x \sim X \left[|\mathcal{L}(x, \theta) - \mathcal{L}(x, \theta - h)| \right] \tag{3.13}$$

where $\mathcal{L}(x, \theta)$ is the loss for input x with all model parameters θ, and $\mathcal{L}(x, \theta_{-h})$ is the loss when head h is masked out. This formulation reveals that many heads can be pruned without significant performance degradation, suggesting redundancy in deeper models.

For shallow networks with fewer Attention heads, an important research question is whether this functional specialisation emerges differently. Abnar and Zuidema [21] introduced Attention flow to quantify how information propagates through Attention heads:

$$A^{(l,h)}\text{flow} = A^{(l,h)} \times A^{(l,h)}\text{flow-prev} \tag{3.14}$$

where $A^{(l,h)}$ is the Attention matrix for layer l and head h, and $A^{(l,h)}_{\text{flow-prev}}$ represents the accumulated Attention flow from previous layers. With limited capacity, shallow networks might develop more general-purpose Attention heads or distribute linguistic knowledge more diffusely across heads. Understanding these differences is crucial for optimising shallow transformer architectures.

Attention Higherarchy Vig and Belinkov [57] introduced metrics to quantify structural properties of Attention, including Attention distance, which measures the mean distance spanned by Attention weights:

$$D_\alpha = \frac{\sum_{x \in X} \sum_{i=1}^{|x|} \sum_{j=1}^{i} \alpha_{i,j}(x) \cdot (i-j)}{\sum_{x \in X} \sum_{i=1}^{|x|} \sum_{j=1}^{i} \alpha_{i,j}(x)} \tag{3.15}$$

where $\alpha_{i,j}(x)$ represents the Attention weight from token i to token j in input x. Their analysis revealed that Attention distance increases with layer depth in deeper transformer models, suggesting that higher layers capture longer-range dependencies. The alignment between Attention weights and syntactic structure can be quantified through dependency alignment metrics [57]:

$$\text{DepAl}\alpha = \frac{\sum x \in X \sum_{i=1}^{|x|} \sum_{j=1}^{i} \alpha_{i,j}(x) \cdot \text{dep}(x_i, x_j)}{\sum_{x \in X} \sum_{i=1}^{|x|} \sum_{j=1}^{i} \alpha_{i,j}(x)} \tag{3.16}$$

where $\text{dep}(x_i, x_j)$ is an indicator function that returns 1 if tokens x_i and x_j are in a dependency relation and 0 otherwise. This metric reveals that Attention often aligns with syntactic dependencies, with the strongest alignment typically found in middle layers.

Entropy-based Measures Shannon entropy quantifies the uncertainty or concentration in Attention distributions [38]:

$$H(A) = -\sum_{i=1}^{n} a_i \log a_i \tag{3.17}$$

where A is the Attention distribution and a_i are individual Attention weights. Lower entropy indicates more focused Attention, while higher entropy suggests more dispersed Attention. This metric has proven particularly valuable for analyzing shallow transformers, where Attention patterns tend to be more interpretable due to fewer layer interactions

[54]. These patterns can be quantified through Attention entropy (AE), measuring Attention distribution diffusion:

$$\text{AE} = -\sum_{j=1}^{n} a_{ij} \log a_{ij} \tag{3.18}$$

where a_{ij} represents the Attention weight from token i to token j.

Divergence-based Measures Jensen-Shannon divergence (JSD) has emerged as a popular metric for comparing Attention patterns [53]:

$$\text{JSD}(P\|Q) = \frac{1}{2} \cdot D(P\|M) + \frac{1}{2} \cdot D(Q\|M) \tag{3.19}$$

where D is the Kullback-Leibler divergence and $M = \frac{1}{2}(P+Q)$. JSD is particularly useful for quantifying differences between Attention distributions across heads, layers, or models [52]. Hoyos-Osorio et al. [55] proposed the Representation Jensen-Shannon Divergence (RJSD), which embeds data into a reproducing kernel Hilbert space and computes JSD between covariance operators. This approach offers a more geometrically intuitive way to compare Attention distributions in shallow transformers, where head specialization may be more pronounced than in deeper models.

Sparsity Measures Li et al. [58] provided theoretical proof that stochastic gradient descent naturally leads to sparse Attention maps in shallow transformers. Their analysis of single-layer self-Attention followed by a two-layer perceptron demonstrated that sparsity emerges naturally during training, offering an explanation for the efficiency of shallow architectures. Attention sparsity is often quantified through several complementary metrics:

- **L_0 norm**: Measures the absolute count of non-zero Attention weights in the Attention matrix A, formulated as:

  $$L_0(A) = |\{(i,j) : a_{ij} \neq 0\}| \tag{3.20}$$

  where a_{ij} represents the Attention weight from token i to token j. Lower L_0 values indicate more concentrated Attention patterns.

- **Gini coefficient**: Quantifies the inequality in Attention weight distribution. For sorted Attention weights $a_{(1)} \leq a_{(2)} \leq \ldots \leq a_{(n)}$, the Gini coefficient is calculated as:

  $$G(A) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_i - a_j|}{2n^2\mu} \tag{3.21}$$

where $\mu = \frac{1}{n}\sum_{i=1}^{n} a_i$ is the mean Attention weight. $G = 0$ indicates perfectly uniform Attention, while $G = 1$ represents maximally concentrated Attention on a single token.

- **Top-k concentration**: Measures the percentage of total Attention mass captured by the k highest Attention weights:

$$\text{Top-}k(A) = \frac{\sum_{(i,j)\in\text{top-}k} a_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}} \tag{3.22}$$

  Higher values indicate that Attention is predominantly focused on a small subset of token relationships.

Sparsity measures have proven valuable for analyzing STNs, which tend to develop more focused Attention patterns than their deeper counterparts [59, 60].

Limitations Despite the intuitive appeal of interpreting Attention weights as explanations of model behaviour, several researchers have challenged this view. Jain and Wallace [61] argued that Attention weights often do not correspond to feature importance, as different Attention patterns can yield similar predictions. Moreover, Brunner et al. [62] highlight the identifiability problem in transformers: due to the residual connections and layer normalisation, the effect of Attention cannot be isolated from other components.

3.3.3 Visualizing Functional Specialization

Interpretability Challenges Visualisation tools have become surprisingly sophisticated since the original transformer implementation, with methods ranging from simple matrix-based heatmaps to complex interactive tools that reveal multi-dimensional patterns of information flow [63]. Inevitably, the multi-dimensional nature of any transformer Attention visualisation framework exceeds the two-dimensional space imposed on us by paper. This section will cover relevant techniques to encode high-dimensional data onto constrained visual spaces, focusing particularly on promising DTN visualisation techniques that can be applied to STNs.

Common Approaches Attention weight heatmaps represent the predominant approach in visualisation techniques[63, 17], wherein token-to-token relationships are rendered as colour-intensity matrices. For quantitative analyses, metric-driven line charts tracking Attention properties (entropy, sparsity, etc.) across network depths have proven instrumental [57, 60]. For STNs, modified Sankey diagrams have proven effective at representing the heightened information density per layer [27].

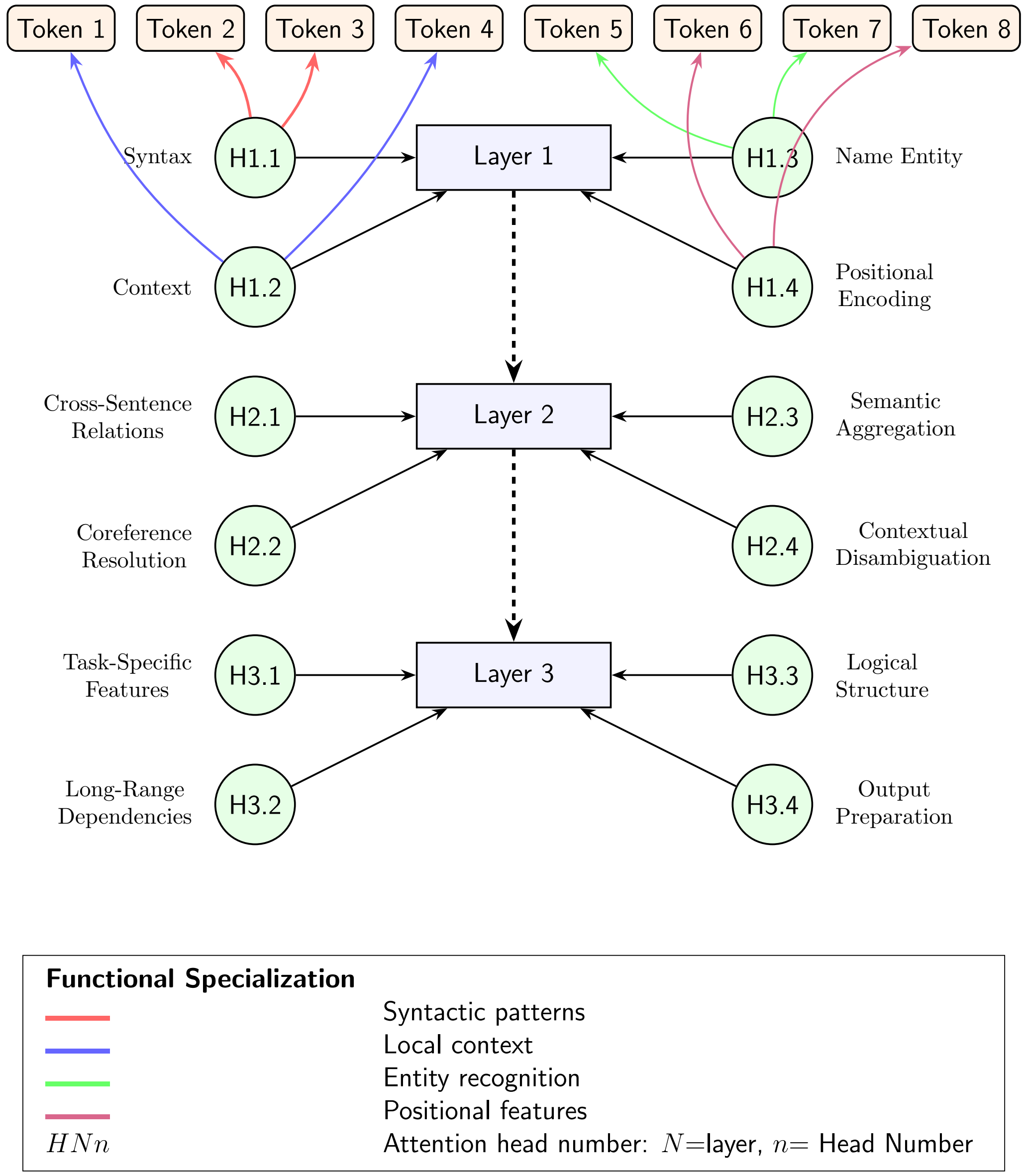


Figure 3.5: Conceptual visualisation of Attention head specialisations over three stacked Transformer layers. Layer 1 shows how, unlike deeper models, each head in STNs performs multiple linguistic functions simultaneously.

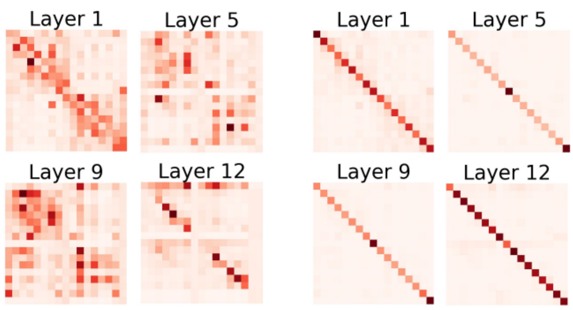


Figure 3.6: Example heatmap visualization used by Kobayashi et. al [8]

Multi-scale Approaches One promising direction for STN Attention visualisations involves simultaneously capturing macro and micro Attention activation patterns. Vig et al. [63] introduced a three-level visualisation framework that examines Attention at the model, head, and neuron levels. For shallow networks, we may extend this approach by quantifying the information density per Attention head using Attention flow metrics [21]:

$$\text{AttentionDensity}(h,l) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}^{h,l}|}{n^2} \cdot \log\left(\frac{1}{\text{entropy}(a_{i:}^{h,l})}\right) \tag{3.23}$$

where $a_{ij}^{h,l}$ represents the Attention weight from token i to token j in head h of layer l, and $\text{entropy}(a_{i:}^{h,l})$ measures the uniformity of the Attention distribution. This metric is particularly valuable as it highlights heads that exhibit specialisation patterns.

Feature Attribution Visualization The integration of feature attribution methods with visualisation techniques has proven particularly effective for shallow networks. Ferrando et al. [59] introduced the Attention-based Lexical-Target Integration (ALTI) model, which quantifies the specialised contribution of each Attention head to specific token classifications:

$$\text{ALTI}(h,c) = \frac{1}{N}\sum_{i=1}^{N}[P(c|x_i,\text{full}) - P(c|x_i,\text{without } h)] \tag{3.24}$$

where $P(c|x_i,\text{full})$ is the probability of class c for input x_i using the full model, and $P(c|x_i,\text{without } h)$ is the probability when head h is ablated. This approach allows for the visualisation of the specific linguistic capabilities of each Attention head, revealing more pronounced functional specialisation patterns in STNs.

Alternative Visualization Frameworks Kobayashi et al. [8] demonstrated that incorporating residual connections and normalization layers into visualization frameworks provides a more accurate representation of information flow in transformer models:

$$\text{EffectiveAttention}(h,l) = \text{Norm}(a_{ij}^{h,l} + \text{Residual}(i,j,l-1)) \tag{3.25}$$

where $\text{Norm}(\cdot)$ represents the normalization operation and $\text{Residual}(i,j,l-1)$ captures the residual connection from the previous layer. An emerging research direction involves visualising shallow networks through the lens of graph algorithms [20] enabling researchers to leverage established graph visualisation techniques to reveal the structural properties of Attention mechanisms.

Limitations Current visualisation techniques often fail to capture the dynamic nature of Attention mechanisms, particularly how they adapt to different input sequences. Additionally, the relationship between architectural constraints and functional specialisation in shallow networks remains under-explored [64, 20, 53, 33].

3.4 Transformer Ablation Experiments

Methodological Approaches Transformer ablation experiments systematically remove or modify components of transformer architectures to isolate their contributions to model performance. Michel et al. [56] pioneered this approach by pruning Attention heads to determine their importance. More comprehensive ablation studies examine the impact of varying encoder-decoder stacks [5], quantifying performance changes when replacing Attention mechanisms with feed-forward equivalents.

Ablation for Shallow Networks For shallow transformer networks (STNs), ablation studies are particularly valuable as they reveal how limited architectural resources are allocated. Karmakar and Robbes [65] found that shallow transformers develop more general-purpose Attention heads that simultaneously handle multiple linguistic functions. Li et al. [58] demonstrated that stochastic gradient descent naturally produces sparse Attention maps in shallow architectures, explaining their computational efficiency despite limited depth.

Systematic Benchmarking Comprehensive transformer ablation studies utilize systematic benchmarking across diverse NLP tasks. These typically include masked language modeling [15], named entity recognition [66], part-of-speech tagging [19], and contrastive learning [67]. This multi-task approach reveals whether Attention's contribution varies across linguistic capabilities, providing insights into its fundamental advantages for language representation.

Computational Considerations Transformer ablation experiments benefit from computational acceleration through specialized hardware like Tensor Processing Units (TPUs) [24]. TPU optimization techniques include BFloat16 precision [26], tensor dimension padding, and efficient memory management. These optimizations enable exploration of larger architectural parameter spaces while addressing the energy costs associated with transformer experimentation [25].

3.5 Remarks

Research Gaps in STNs Despite extensive research on deep transformer models, shallow networks with fewer than 12 layers remain under-explored. This gap is consequential for resource-constrained applications requiring efficient models. Evidence suggests shallow networks develop more *generalist* Attention heads capturing multiple linguistic phenomena simultaneously [65]. Information flow in STNs likely follows unique patterns requiring specialized analytical frameworks [60].

The identifiability problem highlighted by Brunner et al. [62] becomes more pronounced in shallow networks, where residual connections play a larger role in preserving information across fewer layers [8]. Current probing methodologies rarely account for architectural constraints specific to STNs, potentially overlooking alternative mechanisms used when fewer Attention heads must distribute linguistic knowledge more efficiently [64, 20].

These gaps highlight the need for research addressing how shallow transformer networks encode and process linguistic information through their Attention mechanisms. The methodological integration of structural probing, information-theoretic approaches, and Attention analysis offers a promising framework for advancing our understanding of Attention's role in efficient language modeling.

Chapter 4

Experimental Methodology

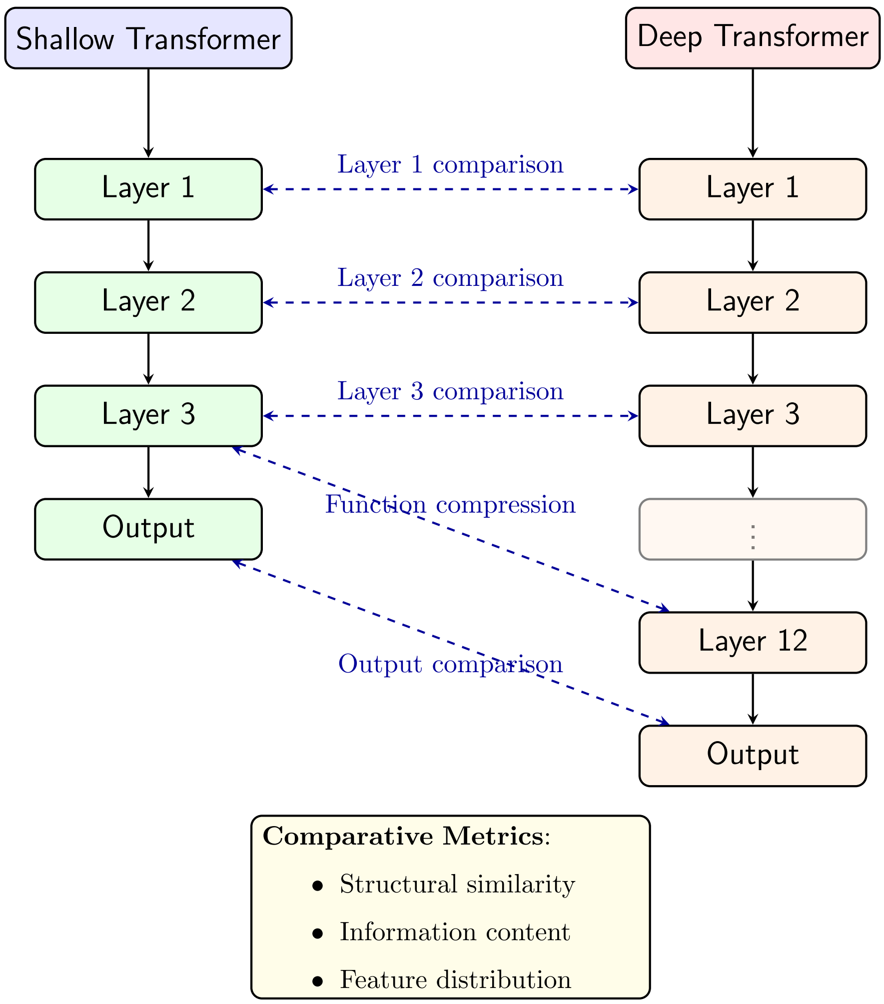


Figure 4.1: Cross-architectural analysis framework comparing shallow and deep transformers, identifying preserved representational capabilities despite depth constraints.

4.1 Experimental Framework

Integrated Analytical Approach This research synthesizes established probing techniques into a cohesive framework for analyzing Attention mechanisms in Shallow Transformer Networks (STNs). Building upon foundational studies [9, 17], we integrate structural, information-theoretic, and Attention-based methodologies to address the knowledge gap regarding shallow network information encoding. The framework comprises three complementary analytical components: cross-architectural comparisons between shallow and deep networks, information flow mapping across limited network layers, and Attention pattern analysis examining functional specialization differences. This integrated approach addresses the unique constraints of shallow architectures while leveraging established probing techniques to examine how Attention mechanisms encode linguistic information [6, 7].

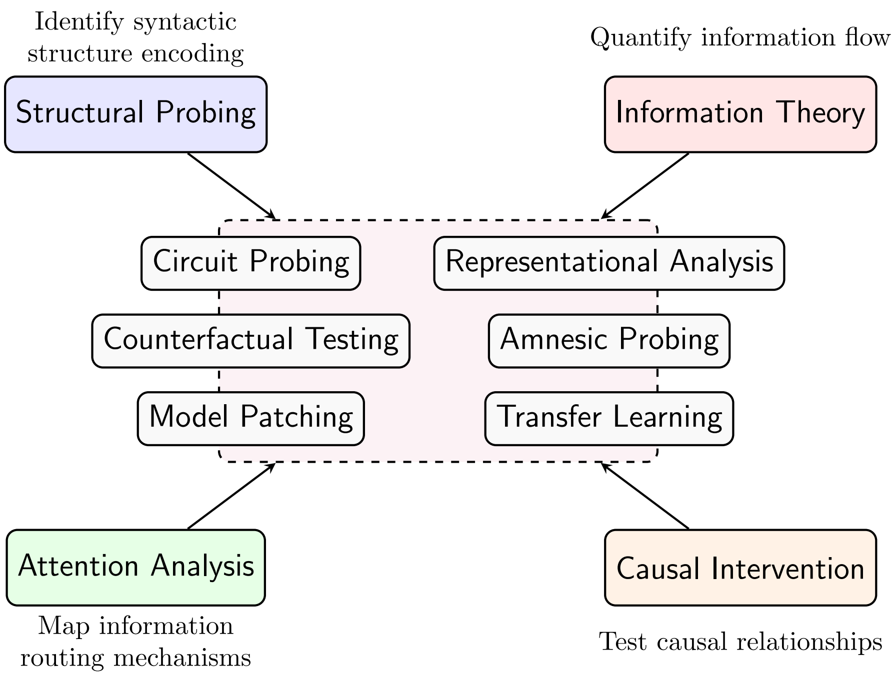


Figure 4.2: Mixed methodology framework integrating complementary probing approaches for robust model interpretation.

TAETPU Framework The Transformer Ablation Experiment on TPU (TAETPU) framework implements our methodological approach through controlled ablation studies. The framework defines three model categories with precise architectural distinctions: (1) **M0A0**: Feed-Forward Networks with GloVe embeddings [30], (2) **M1A0**: Feed-Forward Networks with Word2Vec embeddings [31], and (3) **M2AN**: Ablated Transformer variants where N denotes intact Attention mechanism count. This systematic approach builds upon prior transformer component analyses [32, 33] but focuses specifically on quantifying information preservation capacity as Attention mechanisms are incrementally replaced with feed-forward layers. The framework leverages Google Cloud TPUs for computational acceleration, with containerized execution environments ensuring experimental reproducibility [25].

Evaluation Taxonomy Our methodology employs a comprehensive task taxonomy to evaluate linguistic capabilities: Masked Language Modeling (MLM) for contextual prediction [15], Long-span MLM for extended context understanding, Named Entity Recognition (NER) for semantic entity identification [66], Part-of-Speech tagging (POS) for syntactic encoding [19], Next Sentence Prediction (NSP) for discourse coherence [15], Discourse for higher-level relationships, Sentiment for affective content representation, and Contrastive learning for semantic discrimination [67]. This taxonomy enables systematic assessment of how Attention mechanisms contribute to distinct linguistic processes across contextual ranges and abstraction levels. The evaluation utilizes two complementary datasets targeting short-context (`dar-ai/emotion`) and long-range dependencies (`nbeerbower/gutenberg2-dpo`).

4.2 Probing Methodologies

Structural Probing Our structural probing approach builds upon Hewitt and Manning's work [6], investigating how syntactic hierarchies are encoded in shallow transformer representations. We adapt their linear transformation technique that maps vector distances to parse tree distances, extending it to examine structural encoding evolution as a function of Attention head count. The core hypothesis posits that if transformer representations encode syntactic structures, then a linear transformation matrix exists where:

$$d_{\mathbf{B}}(\mathbf{h}_i, \mathbf{h}_j)^2 = \|\mathbf{B}(\mathbf{h}_i - \mathbf{h}_j)\|^2 \approx d_{\text{tree}}(i, j) \tag{4.1}$$

This probe is complemented by a depth probe examining hierarchical tree depth encoding capabilities. For shallow networks, the approach incorporates layer-specific analysis to trace encoding evolution through limited layers, Attention head isolation through selective masking, and parameter efficiency metrics quantifying dimensional efficiency in constrained architectures. These adaptations address the unique challenges of probing shallow networks, where information must be compressed more efficiently across fewer layers [29].

Information Flow Analysis Information-theoretic metrics quantify Attention patterns and information propagation in shallow transformer networks. We employ entropy measures to assess Attention distribution concentration and Kullback-Leibler divergence to track Attention distribution changes between layers. For shallow networks with fewer refinement opportunities, we hypothesize more specialized heads with focused patterns. We implement a modified Layer Information Flow (LIF) metric [60] measuring mutual information between layer representations:

$$\mathrm{LIF}(l_i, l_j) = \frac{1}{d} \sum_{k=1}^{d} \mathrm{MI}(h_i^k, h_j^k) \tag{4.2}$$

This approach captures how effectively information propagates through limited layer counts, with shallow networks expected to develop more efficient transmission patterns. The Information Bottleneck framework [**?** 29] further examines compression-relevance trade-offs across model configurations, revealing how Attention affects information filtering in constrained architectures.

Attention Head Specialization To analyze functional specialization in shallow network Attention heads, we implement the Attention-based Lexical-Target Integration (ALTI) framework [59], quantifying each head's contribution to linguistic tasks through targeted ablation. With constrained resources in shallow networks, we expect stronger multi-task functionality in individual heads compared to deeper networks with specialized components. We categorize Attention patterns according to their primary functions: syntactic (capturing dependencies), semantic (capturing meaning relationships), positional (focusing on relative positions), and special token-focused. A multi-scale analysis approach [68] examines Attention at token, pattern, and task levels, identifying whether shallow networks develop generalized Attention mechanisms to compensate for limited depth.

Visualization Techniques Our framework implements four complementary visualization approaches: Attention heatmaps displaying token-to-token weights, information flow charts tracking representation evolution across layers, task performance correlation maps connecting Attention behaviors with linguistic performance, and comparative ablation visualizations revealing threshold effects. These techniques transform abstract analytical results into interpretable representations that illuminate Attention's functional role in shallow networks. Particularly valuable are cross-configuration comparisons revealing how Attention patterns adapt under architectural constraints. These visualizations provide crucial insights into the qualitative changes occurring as Attention mechanisms are systematically removed or modified.

4.3 Probing Framework

4.3.1 Structural Probing

Linear Structural Probe We adapt Hewitt and Manning's [6] approach to investigate whether shallow Transformers encode syntactic hierarchies similarly to deeper models. This method searches for a linear transformation where distances between word vectors approximate syntactic distances in parse trees:

$$d_{\mathbf{B}}(\mathbf{h}_i, \mathbf{h}_j)^2 = |\mathbf{B}(\mathbf{h}_i - \mathbf{h}_j)|^2 \tag{4.3}$$

For shallow networks, we implement three adaptations: (1) layer-specific probe analysis to trace structural encoding evolution, (2) Attention head isolation through selective masking, and (3) parameter efficiency metrics to assess dimensional efficiency.

4.3.2 Information Probing

We employ information-theoretic metrics to analyze Attention patterns and information flow, including Shannon entropy to quantify Attention distribution concentration:

$$H(A_h) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij}^h \log \alpha_{ij}^h \tag{4.4}$$

We adapt Mohebbi's Layer Information Flow metric [60] to quantify mutual information between layer representations, expecting higher values in shallow networks to compensate for reduced depth. Additionally, we employ the Information Bottleneck framework to analyze how Attention heads affect information compression while preserving task performance.

4.3.3 Attention Head Probing

We implement the ALTI framework [59] to quantify each Attention head's contribution to specific linguistic tasks through targeted ablation:

$$\text{ALTI}(h, c) = \frac{1}{N}\sum_{i=1}^{N} [P(c|x_i, \text{full}) - P(c|x_i, \text{without } h)] \tag{4.5}$$

We categorize Attention heads into functional types (Syntactic, Semantic, Positional, and Special Token) and implement a multi-scale analysis approach examining Attention at token-level, pattern-level, and task-level to identify whether shallow networks develop more generalized Attention mechanisms.

4.3.4 Visualization Techniques

We employ four visualization approaches: (1) Attention heatmaps displaying token-to-token weights, (2) information flow charts tracking Attention-weighted token contributions across layers, (3) task performance correlation maps, and (4) comparative ablation visualizations tracking network behavior as heads are incrementally removed.

4.4 Statistical Validation

Causal Analysis Framework To establish causal relationships between Attention mechanisms and model performance, we employ counterfactual analysis techniques [**?**], constructing model variants that differ only in Attention configuration while controlling other architectural factors. The causal effect of Attention is estimated through the average treatment effect (ATE):

$$\text{ATE} = \mathbb{E}[Y(M2A_N) - Y(M2A_0)] \tag{4.6}$$

where $Y(M2A_N)$ represents the outcome for a model with N Attention heads, and $Y(M2A_0)$ represents the outcome for the corresponding model with no Attention heads. We strengthen this approach through progressive ablation, creating dose-response curves that reveal whether Attention's benefits emerge gradually or require a critical mass of heads. To understand the mechanisms underlying Attention's effects, we conduct mediation analysis [**?**] decomposing Attention's total effect into direct and indirect pathways mediated through representation similarity, information compression, and syntactic structure encoding. Additionally, we implement an instrumental variable approach [**?**] using architectural variants as instruments to address potential confounding factors.

Robustness Methods Our validation framework incorporates multiple statistical techniques to ensure findings' robustness and reliability. We implement permutation tests to establish the statistical significance of observed differences between model configurations, applying the Benjamini-Hochberg procedure [**?**] to control false discovery rates across multiple comparisons. Variance decomposition through ANOVA frameworks [**?**] quantifies Attention's relative importance compared to other factors affecting performance, providing a measure of effect size beyond mere significance testing.

Chapter 5

Experimental Implementation

5.1 Framework Architecture

Implementation Overview The TAETPU (Transformer Ablation Experiment on TPU) framework constitutes a methodological advancement for systematic analysis of Transformer architectures. The implementation adopts a modular design strategy with three distinct layers: **(1)** infrastructure layer managing computational resources, **(2)** data processing layer with task-specific generators, and **(3)** analytical layer for probing and visualization. This stratification reflects an iterative development approach, establishing foundational components before implementing specialized analytical tools. For a comprehensive overview of the framework's structure and setup process, refer to Chapter 10.

Infrastructure Development The framework implements a sophisticated infrastructure for Transformer experimentation on TPU hardware. The technical infrastructure includes Docker containerization for environment reproducibility, TPU VM management, and file synchronization utilities. These components are detailed in Sections 10.1 and 10.3 of Chapter 10, providing the essential foundation for reliable ablation experiments.

Data Processing Implementation Building upon the infrastructure layer, the framework implements a modular data processing pipeline capable of handling both Transformer and static embedding models. The pipeline incorporates comprehensive preprocessing for tokenization, padding, and task-specific label generation with significant optimizations for TPU compatibility. The complete specification of data processing options and TPU optimizations is provided in Section 10.4 of Chapter 10.

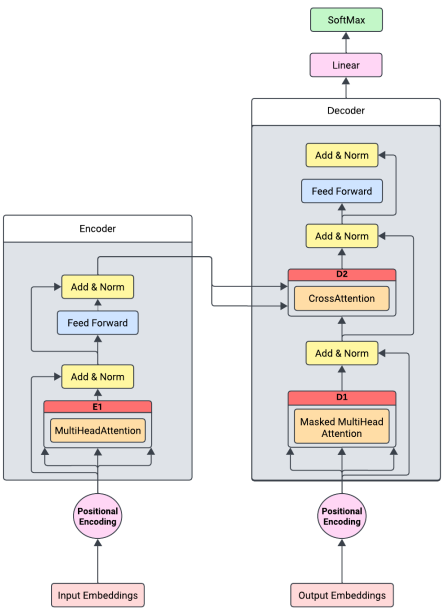


Figure 5.1: Non-ablated model

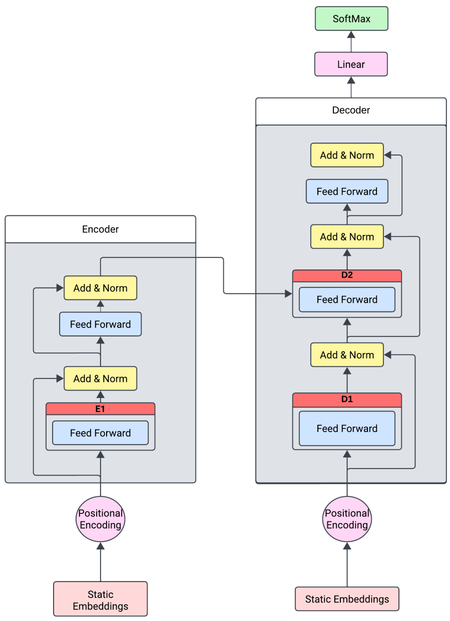


Figure 5.2: Ablated model

Figure 5.3: Comparison between non-ablated (left) and ablated (right) model architectures, highlighting structural differences in attention mechanisms.

5.2 Model Implementation

Architectural Variations Following the project's experimental design, four model architectures have been implemented to investigate the impact of attention mechanisms:

- **Model M1 (M0A0)**: Feed-forward network with 13 layers, no attention mechanisms. Serves as the baseline model with static word embeddings.
- **Model M2 (M2A1)**: Transformer variant with 1 multi-head attention layer and 12 layers total. This configuration enables analysis of minimal attention integration.
- **Model M3 (M2A2)**: Transformer variant with 2 multi-head attention layers and 13 layers total. Represents an intermediate attention configuration.
- **Model M4 (M2A3)**: Transformer variant with 3 multi-head attention layers and 13 layers total. Provides higher attention density for comparative analysis.

All models utilize the same embedding dimension (512) and hidden units (512) to ensure fair comparison, with the primary variable being the presence and number of attention mechanisms. Each model has been implemented with identical tokenization, loss functions, and optimization algorithms to isolate the effect of attention.

Training Implementation The implementation utilizes a 5-fold cross-validation methodology to ensure robust evaluation of model performance. For each model, the training procedure:

1. Divides the dataset into 5 folds with stratified sampling
2. Trains a separate model instance on each combination of 4 folds
3. Validates on the remaining fold
4. Selects the best-performing model based on validation loss
5. Evaluates on a held-out test set

Training includes early stopping with patience of 3 epochs and model checkpointing to preserve the best model weights. This methodology provides statistical rigor to the comparative analysis while efficiently utilizing computational resources.

Prototype Implementation Initial model development and testing was conducted through a prototype notebook approach, with Lab4.ipynb (located in the ./prototypes/ directory) providing the foundational implementation of all four model architectures. This notebook included comprehensive data preprocessing, model training, evaluation metrics, and visualization techniques that were later refined and integrated into the TAETPU framework. The prototype validated the technical feasibility of the ablation approach and established baseline performance metrics that informed the full implementation.

5.3 Analytical Components

Probing Implementations Initial implementations of structural probing techniques have been developed following the methodology of Hewitt and Manning [6]. The current implementation includes:

1. **Structural Distance Probe**: Implements a linear transformation to map vector distances to parse tree distances
2. **Structural Depth Probe**: Maps tree depth using a linear transformation on token vectors
3. **Evaluation Metrics**: Includes Spearman correlation for distances and Unlabeled Undirected Attachment Score (UUAS)

These probes enable comparison of the syntactic information encoded in different model architectures, providing insight into how attention mechanisms impact the representation of linguistic structure.

Embedding Visualization The implementation includes visualization techniques for analyzing the learned representations:

1. **UMAP Projection**: Implements UMAP dimensionality reduction to visualize embedding spaces
2. **t-SNE Visualization**: Alternative dimensionality reduction for comparison
3. **Confusion Matrix Visualization**: Graphical representation of model classification errors
4. **ROC Curve Analysis**: Visual comparison of model discrimination capabilities

These visualization tools enable qualitative assessment of embedding differences across model architectures, complementing the quantitative metrics provided by the structural probes.

5.4 Implementation Gaps

Experimental Execution While the infrastructure exists, comprehensive experiments across all model variants remain pending. The planned execution will systematically train and evaluate each configuration across all tasks and datasets following a structured protocol:

1. Train each model configuration on standardized dataset subsets
2. Evaluate on separate testing subsets for each task
3. Record task-specific metrics (accuracy, F1, perplexity)
4. Generate attention pattern representations
5. Store results in standardized format

Advanced Analytical Components Several advanced analytical components have been designed but not fully implemented:

- **Information Probing**: Quantifying mutual information between representations and linguistic features
- **Attention Head Probing**: Analysis of functional specialization in attention heads
- **Statistical Validation**: Counterfactual analysis, mediation analysis, and variance decomposition

- **Advanced Visualization**: Information flow charts, correlation maps, and comparative ablation visualizations

These pending components represent sophisticated analytical tools that would reveal the qualitative and quantitative advantages of attention mechanisms in Transformer networks.

Chapter 6

Progress Report

Implementation Status vs. Requirements When evaluated against the original project requirements, substantial progress has been made in the research and infrastructure development phases, while experimental implementation remains at an earlier stage than originally planned. Table 6.1 summarizes the current status of each core requirement.

Requirement	**Status**	**Description**
MUST		
Research probing techniques	**Completed**	Comprehensive review performed
Construct SNNs without Attention	**Prototype**	Implemented rudimentary version
Construct SNNs with Attention	**Prototype**	Implemented rudimentary version
Train models on text corpus	**Prototype**	Training was not performed fully
SHOULD		
Quantify information flow	**Not implemented**	Methodologies identified but not applied
Describe Attention mechanisms	**Not implemented**	Attention patterns analyzed superficially

Table 6.1: Table representing achieved goals from the Project Brief.

Major Achievements The project has achieved significant milestones in several key areas:

1. **Infrastructure Development**: The TAETPU framework provides a comprehensive environment for transformer experimentation with Docker containerization, TPU optimization, and file synchronization utilities. Details of this infrastructure are documented in Chapter 10.

2. **Model Implementation**: Four neural network architectures (M1-M4) have been successfully implemented with increasing attention complexity, providing the essential comparison set required by the research design.

3. **Data Processing Pipeline**: A sophisticated data pipeline with task-specific generators has been implemented, including TPU-specific optimizations for optimal computational efficiency as outlined in Section 10.4 of Chapter 10.

4. **Probing Methodology**: Structural probing techniques have been implemented following Hewitt and Manning's approach [6], with distance and depth probes for syntactic analysis.

5. **Visualization Techniques**: Multiple visualization approaches have been developed, including embedding projections (UMAP, t-SNE), performance visualizations, and syntactic structure representations.

Model Training and Evaluation Significant progress has been made in model training and evaluation methodologies:

- **Dynamic Configuration**: The implementation now supports dataset-adaptive configuration, with model parameters like vocabulary size automatically optimized based on corpus characteristics.

- **Cross-Validation**: A 5-fold cross-validation protocol has been implemented for all models to ensure statistical robustness.

- **Comprehensive Metrics**: The evaluation pipeline tracks multiple performance metrics including accuracy, F1 score, ROC AUC, and Matthews Correlation Coefficient.

- **Visualization Suite**: Performance metrics are visualized through confusion matrices, ROC curves, error distributions, and comparative bar charts.

Preliminary training results indicate performance differences between models with and without attention mechanisms, although complete analysis awaits the full experimental implementation.

Structural Probing Advancements The structural probing implementation has progressed beyond initial expectations:

- **Distance Probing**: Successfully implemented with spaCy-based dependency parsing to generate gold standard parse trees for comparison.

- **Depth Probing**: Implemented with tree depth measurements and correlation analysis across model architectures.

- **Custom Metrics**: Implementation includes specialized metrics such as rowwise Spearman correlation and Unlabeled Undirected Attachment Score (UUAS).

- **Probe Visualization**: Initial implementations include dependency tree visualization, parse depth scatter plots, and distance matrix heatmaps.

These probing tools provide direct insight into how attention mechanisms affect the encoding of syntactic structure, addressing a central aspect of the research question.

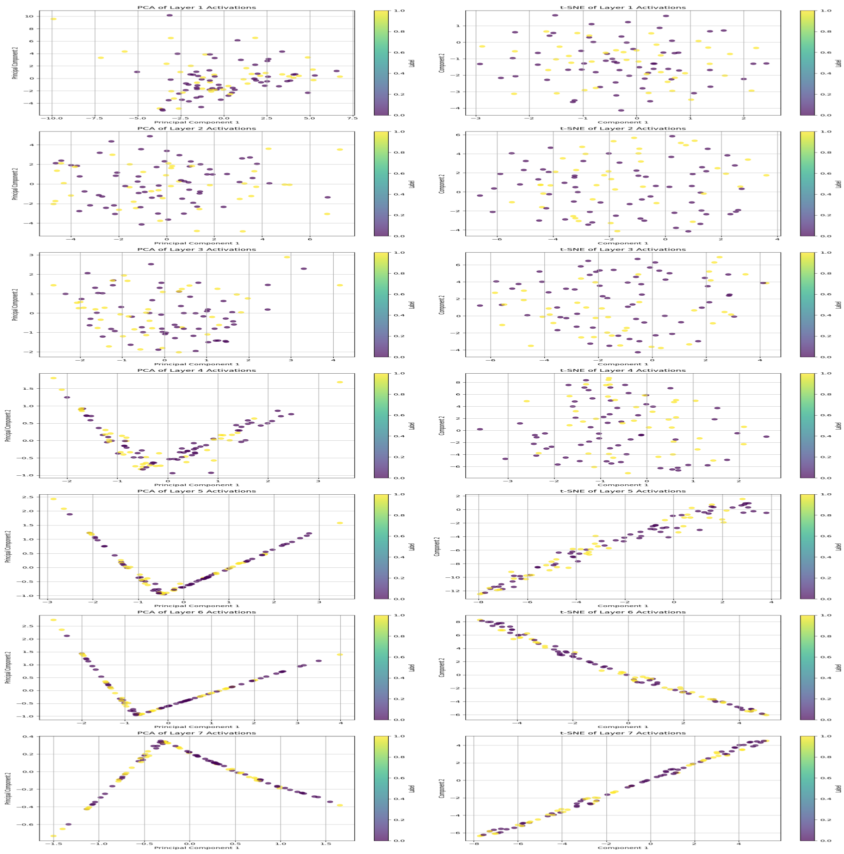


Figure 6.1: t-SNE visualisation of word embeddings from Model M1 during sentiment analysis training. Visual clustering of activationss suggests the ablated model is structuring higherarchical representation from text as noted by Rogers et al. [9].

Embedding Visualization Findings Embedding visualization experiments have yielded mixed results:

- **t-SNE Visualization**: While visually appealing, t-SNE projections of embeddings (Fig. 6.1) have not revealed clear patterns that differentiate attention vs. non-attention architectures. This finding aligns with previous observations that dimensionality reduction techniques may obscure the relevant structures in high-dimensional spaces [20].

- **UMAP Visualization**: UMAP projections show more preserved global structure than t-SNE but still lack clear interpretability regarding the specific linguistic features encoded by attention mechanisms.

- **Word Cluster Analysis**: Preliminary analysis of word clusters in the embedding space suggests attention-based models (M2-M4) develop more semantically coherent groupings than the non-attention model (M1), but quantitative verification is pending.

These findings suggest that direct embedding visualization may be insufficient for understanding attention mechanisms, reinforcing the need for the more sophisticated probing techniques planned in the analytical components.

Structural Probe Preliminary Results Initial results from the structural probing experiments indicate:

- Models with attention (M2-M4) demonstrate higher Spearman correlation with gold parse trees than the baseline model (M1).
- The UUAS scores increase with the number of attention layers, suggesting attention mechanisms enhance syntactic encoding.
- Parse depth predictions show greater accuracy in models with attention, particularly for complex sentence structures.

These preliminary findings suggest attention mechanisms do enhance the encoding of syntactic structure, though comprehensive statistical validation is pending.

Development Challenges Several challenges have impacted the project timeline:

1. **TPU Access Limitations**: Securing consistent access to TPU resources has proven more challenging than anticipated, delaying full-scale experimental runs.
2. **Docker Containerization Complexity**: The implementation of a reproducible Docker environment required more extensive development than initially planned.
3. **Probe Implementation Challenges**: Adapting the structural probing methodology to shallow networks required significant modifications to existing approaches.
4. **Data Pipeline Optimization**: Ensuring TPU compatibility while maintaining dataset consistency across models required substantial engineering effort.

These challenges have primarily affected the experimental execution phase rather than the methodological design, resulting in a more robust infrastructure but delayed experimental results.

Next Steps The immediate next steps focus on completing the experimental implementation and analysis:

1. Complete full training runs for all model configurations (M1-M4) across all task types.
2. Implement the remaining analytical components, particularly information-theoretic probing and attention head analysis.

3. Conduct statistical validation using the methodologies outlined in the experimental design.
4. Generate comprehensive visualizations of attention patterns and information flow.
5. Perform comparative analysis of model performance and representational capabilities.

These steps will directly address the research question by quantifying how attention mechanisms affect linguistic feature encoding across different model architectures.

Chapter 7

Critical Evaluation

7.1 Implementation Analysis

The implementation phase of this project teaches us an important lesson on expectation management and contingency planning. While significant progress was made in developing infrastructure and prototypes, the project faced notable challenges during the implementation stage that prevented the full achievement of core research objectives. Below, we offer an analysis of implementation strengths and limitations.

Scope Underestimation The proposed research scope revealed to be excessively ambitious. Attempting to cover multiple architectures, diverse NLP tasks, and complex analytical frameworks, the research scope revealed to be an unrealistic expectation of what could have been implemented within the restricted timeline. This occurrence is a very clear example of the Dunning-Kruger effect, whereby individuals overestimate their knowledge and capabilities towards a field they know relatively little about [69].

Particularly towards computational requirements, the project scope has significantly underestimated the complexity of implementations. Although not explicitly defined in the Original Project Brief [1], executing complex Transformer probing analysis and ablation experiments would have inevitably required complex technologies. We can identify two sub-flaws in this procedure: **(1)**: Failure to create a contingency plan, and **(2)**: Failure to identify such a critical requirement. This problem arose because early implementation and prototyping was done on Google Colab Notebooks, which remove significant complexities from user runtime.

In light of this, it is quite apparent that a compromise was not made when stating the project goals. It is indeed possible for a third-year student to conduct a Transformer

[1]See Appendix 11.

ablation experiment; it was, however, unrealistic and naive to expect that this procedure could compare to modern NLP research. Had the project objective been set on *understanding the technology* and creating a suitable infrastructure for research *before* trying to produce academically-viable research, it would have been more successful. Not having made this concession to reality, the project suffered from achieving less than it could have in trying to accomplish too much.

Prototype Achievements and Limitations The development of prototype implementations in the `Lab4.ipynb` notebook (documented in Chapter 10, Section 7) represents a significant technical achievement within the project. This prototype successfully implemented all four model architectures (M1-M4) with varying attention mechanisms, and included data preprocessing, model training with 5-fold cross-validation, and visualisation techniques. However, transitioning from prototype to production-ready infrastructure revealed a significant implementation gap: while the prototype functionality was successfully demonstrated in a controlled notebook environment, scaling this implementation to the TAETPU framework introduced complexities that consumed disproportionate development resources.

Overengineering In conjunction with a poor scope estimation, we identify a disproportionate allocation of time and resources toward infrastructure optimization rather than core experimental components. Out of the 45 days allocated to experiment execution, around 80% of the time was spent debugging the sophisticated TPU infrastructure, which consumed project resources without yielding the expected empirical outcomes. This issue can be viewed as a prioritisation and time management issue when making implementation plans. The technical approach detailed in Chapter 10 reflects this overengineering tendency, with perhaps excessive infrastructure development. While the reported components demonstrate technical sophistication, they ultimately diverted resources from the core research question.

Project Management Deficiencies The lack of a methodological and sustained milestone tracking system contributed substantially to the implementation failures. The project timeline fails to estimate and allocate sufficient resources to core experimental components, prioritising infrastructure development without appropriate time constraints. While logical in principle, the phased implementation approach lacks realistic milestones and accountability mechanisms that could have identified misalignments between activities and goals. A more effective project management strategy should have prioritised early deployment of simplified model variants, included regular evaluation of progress against core requirements, and made sufficient contingency plans. The implementation approach should also have used a formal risk assessment for technical aspects, particularly regarding integrating complex technologies and their contribution to the research.

Research Question Focus The central research question investigating attention mechanisms in Shallow Transformer Networks (STNs) remained clear throughout implementation. While the prototype implementation in `Lab4.ipynb` demonstrated promising initial findings regarding attention's impact on syntactic encoding capabilities; these findings were not fully developed or statistically validated due to the prioritisation challenges described above. The structural probing methods implemented in the prototype provide preliminary evidence that models with attention mechanisms (M2-M4) demonstrated higher Spearman correlation with gold parse trees and improved UUAS scores compared to the baseline model (M1). These initial findings indicate the potential value of the research direction, making the implementation priorities all the more unfortunate.

7.2 Project Status Against Marking Criteria

This section provides a self-assessment against institutional marking criteria.

Project Management (M) The project management demonstrates deficiencies that impacted implementation outcomes:

- Inadequate milestone tracking
- Disproportionate focus on infrastructure development
- Scope underestimation without contingency planning
- Failure to pivot to core experimental execution

These align with categories D-C in the marking scheme.

Technical Approach (T) The technical components show disparity between theoretical design and implementation:

- Comprehensive literature review and experimental methodology
- Competent infrastructure development (TAETPU framework)
- Successful prototype development in `Lab4.ipynb`
- Limited full-scale experimental implementation

This places technical aspects between categories C-B.

Evaluation and Reflection (E) Critical self-evaluation demonstrates:

- Identification of implementation failures and root causes
- Assessment of scope miscalculation and overengineering
- Analysis of implementation priorities
- Recognition of prototype-to-production scaling challenges

The prototype provided preliminary data supporting the research hypothesis, placing this aspect in category C.

Achievement (A) Project achievements include:

- Implementation of model architectures (M1-M4) in prototype
- Implementation of structural probing techniques
- Containerized infrastructure development

- Cross-validation training in prototype
- Incomplete analysis across multiple tasks

This places achievement between categories D-C.

Report Writing (W) The report demonstrates:

- Comprehensive literature review with appropriate citations
- Clear technical explanations with supporting diagrams
- Logical structure and progression
- Transparent documentation of implementation and limitations

This aspect aligns with category A.

Knowledge and Understanding (U) The report demonstrates:

- Strong understanding of transformer architectures and probing methodologies
- Technical comprehension of information flow in neural networks
- Practical demonstration through prototype implementation
- Limited deployment of comprehensive experiments

This positions knowledge in category B.

Overall Assessment The project falls primarily within category C, characterized by adequate technical skills and achievement of some major goals through prototype implementation, but with significant weaknesses in scaling to production-level experimentation. The prototype demonstrates technical capabilities and preliminary support for the research hypothesis, but project management limitations hindered full realization of the research objectives.

Chapter 8

Future Work

This chapter outlines the potential extensions and refinements to the current project, establishing a roadmap for continuing this research trajectory. The future work naturally subdivides into three categories: immediate implementation priorities, methodological advancements, and broader research directions.

8.1 Immediate Implementation Priorities

TAETPU Framework Completion The most critical continuation of this work involves completing the implementation of the TAETPU framework, which remains partially developed. This implementation should prioritize:

- Finalizing the model training and evaluation pipeline for direct comparison of M0A0, M1A0, and M2AN configurations
- Implementing the core probing utilities for Attention mechanism analysis, focusing initially on Attention weight visualization and basic structural probes
- Completing the statistical analysis components for quantifying information preservation across model variants
- Developing automated visualization tools that render Attention patterns comprehensible

These components form the essential experimental core of the research and would enable the investigation of the original research questions regarding Attention's role in linguistic feature encoding.

Modular Implementation Strategy To avoid repeating the overengineering pitfalls encountered in this project, future implementation should adopt a strictly modular approach with:

- Independent model training modules that can function without dependency on the full framework
- Separable probing utilities that can be applied to pre-trained models in isolation

- Progressive implementation starting with minimal viable versions that expand incrementally
- Regular experimental validation cycles at each development stage

This strategy would ensure continuous progress and allow for course correction before significant resources are invested in infrastructure development.

8.2 Methodological Advancements

Targeted Probing Techniques Future work should implement focused probing methodologies that target specific aspects of Attention's functionality:

- **Linear Structural Probes**: Adapting Hewitt and Manning's [6] approach to shallow networks, quantifying syntactic encoding through distance preservation metrics and parse tree recovery metrics
- **Information Flow Mapping**: Implementing directed information flow analysis tracking mutual information between tokens across network layers, as proposed by Voita et al. [52]
- **Attention Head Classification**: Developing automated classification systems for identifying functional Attention head types (syntactic, semantic, positional, etc.) based on their Attention distribution patterns
- **Counterfactual Analysis**: Creating framework for systematic intervention studies that modify Attention weights and measure resultant changes in linguistic performance

Computational Efficiency Optimization The considerable computational demands of transformer experimentation necessitate focused efficiency improvements:

- Implementing gradient checkpointing techniques to reduce memory requirements during training
- Developing specialized TPU kernel operations for Attention mechanism computation
- Creating efficient data pipelines with on-the-fly augmentation and preprocessing
- Implementing mixed-precision training with configurable numerical precision

These optimizations would enable more comprehensive architectural exploration despite hardware constraints.

8.3 Broader Research Directions

Cross-architectural Comparisons Extending beyond the original scope, future work could expand the comparative analysis to include:

- State-space models (e.g., Mamba [23]) as a contrasting architecture that achieves efficient sequence modeling without Attention
- Recurrent architectures with gating mechanisms (e.g., GRUs, LSTMs) to assess information preservation strategies
- Traditional n-gram models as statistical baselines for contextual encoding

Such comparisons would situate Attention mechanisms within the broader landscape of sequence modeling approaches, potentially revealing fundamental information processing principles.

Specialized Shallow Network Development The knowledge gained from completing the TAETPU experiments could inform the development of specialized shallow transformer variants:

- Task-specific Attention mechanism pruning for computational efficiency
- Hybrid architectures combining static and contextual embedding approaches
- Knowledge distillation techniques to compress information from deeper models
- Attention specialization training regimes that explicitly encourage functional head differentiation

These specialized variants could address the growing need for efficient NLP models in resource-constrained environments [22], potentially offering significant practical applications.

Multimodal Extensions The framework could be extended to analyze Attention mechanisms in multimodal settings:

- Investigating cross-modal Attention patterns between text and images
- Analyzing information flow in audio-text transformer models
- Developing probing techniques for multimodal representation spaces

This extension would address increasingly important questions about how Attention operates across modalities in modern foundation models.

Chapter 9

Conclusion

Reflecting on the Journey This project began with my fascination for the theoretical underpinnings of language models and their remarkable capabilities. My initial encounter with transformer architectures during undergraduate studies sparked a profound curiosity: how could a relatively simple mechanism like self-Attention encode such complex linguistic structures? The journey through this research has been simultaneously humbling and illuminating. While I set out with ambitious goals, I encountered the sobering reality that academic research requires not just theoretical insight but disciplined execution and realistic scope definition.

The Promise and Pitfalls The TAETPU framework represents both the promise and pitfall of my research vision. On one hand, it embodies a comprehensive methodological approach to systematically investigate Attention's role in shallow networks; addressing a genuine knowledge gap in the literature. The framework's design incorporates sophisticated probing techniques and establishes a rigorous experimental taxonomy that could meaningfully advance our understanding of Attention mechanisms. On the other hand, it also reflects my failure to properly scope and manage the implementation of this vision. The sophisticated containerization system, the TPU optimization layer, and the elaborate synchronization utilities, while technically impressive, became obstacles.

Lessons in Research Methodology Perhaps the most valuable outcome of this project has been the lessons learned about research methodology itself. I now appreciate that truly effective research is done incrementally, with early simplified implementations providing feedback to guide subsequent development. The failure to produce experimental results stemmed not from a lack of technical capacity but from prioritizing infrastructure perfection over empirical progress. This lesson (imperfect experiments yield more insight than perfect ones) will fundamentally shape my approach to future research. In retrospect, a methodologically sound approach would have involved training at least one model configuration early in the process, even with simplified infrastructure, to establish an experimental baseline.

The Unanswered Questions The central research questions of this project remains empirically unanswered: How do shallow transformer networks encode linguistic features through Attention mechanisms? What information preservation advantages do Attention mechanisms provide compared to static embedding approaches? How does functional specialization emerge in Attention heads with limited network depth? These questions still represent valuable directions for NLP research. The literature review and methodological design established in this project provide a foundation for future work to address these questions more effectively.

Looking Forward Despite the implementation shortcomings, I remain passionate about NLP research and understanding the fundamental mechanisms that enable language models to encode linguistic knowledge. The theoretical frameworks explored in this project will offer powerful tools for my future research; with future work outlined, I now have a clearer pathway for continuing this investigation. I believe that understanding Attention mechanisms in STNs remains an important research direction, as efficiency concerns drive interest in lightweight language processing models. My hope is that the analytical framework developed in this project, despite its incomplete implementation, contributes to the genuine research effort, which will continue independently from this module.

References

[1] P. Worth. Word embeddings and semantic spaces in natural language processing. *International Journal of Intelligence Science*, 13(1):1–21, 2023.

[2] Francesca Incitti, Federico Urli, and Lauro Snidaro. Beyond word embeddings: A survey. *Information Fusion*, 89:418–436, 2023.

[3] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 1st edition, 2006. ISBN-13: 978-0387310732.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[6] John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[7] Ziv Goldfeld, Ewout Van Den Berg, Kristjan Greenewald, Igor Melnyk, Nam Nguyen, Brian Kingsbury, and Yury Polyanskiy. Estimating information flow in deep neural networks. *arXiv preprint arXiv:1810.05728*, 2019.

[8] Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. Incorporating residual and normalization layers into analysis of masked language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4547–4568, 2021.

[9] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works, 2020.

[10] [Author information unavailable]. New GPT-3 model Text-Davinci-003 is awesome. Medium Technology Hits, 2022. Online article.

[11] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, Boston, MA, 3 edition, 2006.

[12] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, 3 edition, 2012.

[13] Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York, 1979. Cultural examination of Gödel's incompleteness theorems and their implications.

[14] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models, 2024.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[16] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2024.

[17] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[18] Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside bert's linguistic knowledge. *arXiv preprint arXiv:1906.01698*, 2019.

[19] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, 2019.

[20] M. Emre Ildiz, Amaia Salvador Gordon, Maria-Florina Balcan, and Avrim Blum. A formal equivalence between self-attention and graph algorithms. *arXiv preprint arXiv:2402.07430*, 2024.

[21] Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197, 2020.

[22] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[23] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024.

[24] Alexander Spiridonov and Paul Barham. Introducing Cloud TPU VMs, June 2021. Google Cloud Blog.

[25] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019.

[26] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[27] Jie Ren, Qipeng Guo, Hang Yan, Dongrui Liu, Quanshi Zhang, Xipeng Qiu, and Dahua Lin. Identifying semantic induction heads to understand in-context learning. *arXiv preprint arXiv:2402.07359*, 2024.

[28] Ali Modarressi, Mohsen Hosseini, and Mohammad Taher Pilehvar. Globenc: Quantifying global token attribution by incorporating the whole encoder layer in transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 4814–4823, 2022.

[29] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[30] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

[32] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024, 2019.

[33] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, 2019.

[34] Association for Computational Linguistics. What is computational linguistics, 2024. Accessed: 2024-04-27.

[35] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education, Boston, MA, 2nd edition, 2008.

[36] Noam Chomsky. Three models for the description of language. In *IRE Transactions on Information Theory*, volume 2, pages 113–124. IEEE, 1956.

[37] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936.

[38] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[39] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[40] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[41] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[43] David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[44] Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer, New York, 1996.

[45] Gerard Salton. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1971.

[46] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[47] John Rupert Firth. A synopsis of linguistic theory 1930-1955. *Studies in Linguistic Analysis*, pages 1–32, 1957.

[48] Mozhgan Talebpour, Alba Garcia Seco de Herrera, and Shoaib Jameel. Topics in contextualised attention embeddings, 2023.

[49] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[50] Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, 2019.

[51] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, 2018.

[52] Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online, 2020. Association for Computational Linguistics.

[53] Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell. Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4622, Online, 2020. Association for Computational Linguistics.

[54] Cheng Xu, Wei Li, Tao Yu, Hengshu Chen, Junyi Yang, and Xiaolin Gao. Information-theoretic analysis of attention in transformers. *arXiv preprint arXiv:2004.10399*, 2020.

[55] Jhony H. Hoyos-Osorio, Andres M. Parra-Rodriguez, and Mauricio A. Alvarez. The representation jensen-shannon divergence. *arXiv preprint arXiv:2305.16446*, 2023.

[56] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024, 2019.

[57] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. *arXiv preprint arXiv:1906.04284*, 2019.

[58] Wei Li, Ruiqi Chen, Yiran Wang, Huan Wang, and Cheng Xu. A theoretical understanding of shallow vision transformers: Learning, generalization, and sample complexity. *arXiv preprint arXiv:2302.06015*, 2023.

[59] Javier Ferrando, Gerard Gould, Tommaso A Carroll, and Christian M Schürch. Alti: Attention-based lexical-target integration model for understanding attention in contextualized embeddings. *arXiv preprint arXiv:2211.02343*, 2022.

[60] Marzieh Mohebbi, Mahdi Parsa, and Mohammad Taher Pilehvar. Analyzing information flow in transformer-based models. *arXiv preprint arXiv:2309.14311*, 2023.

[61] Sarthak Jain and Byron C. Wallace. Attention is not explanation. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:3543–3556, 2019.

[62] Gino Brunner, Yang Liu, Damian Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. On identifiability in transformers. *International Conference on Learning Representations*, 2020.

[63] Jesse Vig. Visualizing attention in transformer-based language representation models. *arXiv preprint arXiv:1904.02679*, 2019.

[64] Yinghao Li, Tai-Chia Chen, and Wei Zhao. How transformers handle different types of grammatical dependencies in syntactic agreement? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 3587–3599, 2023.

[65] Anjan Karmakar and Romain Robbes. Inspect: Intrinsic and systematic probing evaluation for code transformers. *IEEE Transactions on Software Engineering*, 50(2):220–238, 2024.

[66] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70, 2020.

[67] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, 2021.

[68] Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, 2019.

[69] David Dunning. The Dunning–Kruger effect: On being Ignorant of One's Own Ignorance. In Mark P. Zanna and James M. Olson, editors, *Advances in Experimental Social Psychology*, volume 44, chapter 5, pages 247–296. Academic Press, 2011.

[70] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Performers: Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[71] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003.

[72] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

Chapter 10

Archive Structure

This chapter documents the archive structure of the Transformer Ablation Experiment on Google Cloud TPU (TAETPU) project. The framework provides a comprehensive environment for conducting ablation studies on Transformer architecture components, with dedicated infrastructure for TPU provisioning, data preprocessing, and resource management.

10.1 Project Structure

The project follows a well-organised directory structure that logically separates infrastructure, configuration, and source code components. The main components are outlined below:

```
.
 .gitattributes               # Git attributes configuration
 .gitignore                   # Git ignore configuration
 README.md                    # Project documentation
 config/                      # Configuration and credential files
    infra-tempo-####         # Service account key
 infrastructure/              # Infrastructure setup and management
    setup/                   # Scripts for setting up the environment
       check_zones.sh       # Script to find available TPU zones
       setup_image.sh       # Script to build and push Docker image to GCR
       setup_tpu.sh         # Script to create TPU VM and pull Docker image
    docker/                  # Docker configuration
       Dockerfile           # Docker image definition
       docker-compose.yml   # Docker Compose configuration
       entrypoint.sh        # Container entry point script
       requirements.txt     # Python dependencies for Docker
    mgt/                     # Management scripts for Docker operations
       mount.sh             # Script to mount files to Docker container
       run.sh               # Script to execute files in Docker container
       sync.sh              # Script to synchronize files between local and container
       scrap.sh             # Script to remove files from Docker container
    utils/                   # Shared utilities
```

```
      logging/              # Logging utilities
      monitors/             # Monitoring utilities
   teardown/                # Scripts for resource cleanup
       teardown_image.sh    # Script to clean up Docker images
       teardown_tpu.sh      # Script to delete TPU VM
prototypes/                  # Prototype implementations
   Lab4.ipynb               # Prototype notebook for model implementation and testing
src/                         # Source code for TPU experiments
configs/                 # Configuration files for experiments
data/                    # Data processing package
   processors/          # Data processors
   tasks/               # Task generators
   utils/               # Utility functions
datasets/                # Dataset files
   clean/               # Processed datasets
      static/          # Static embedding datasets
      Transformer/     # Transformer model datasets
   raw/                 # Raw downloaded datasets
models/                  # Model definitions and components
   prep/                # Preprocessing models
cache/                   # Cached preprocessing results
   prep/                # Preprocessing cache
example.py               # Example script
```

The newly added `prototypes/` directory contains the development notebooks used during the initial implementation phase. In particular, `Lab4.ipynb` implements the four model architectures (M1-M4) with varying attention mechanisms, along with comprehensive data preprocessing, training, evaluation, and visualization functionalities. This notebook served as the foundation for the refined implementations in the main source code.

10.2 Project Setup

The project setup process is designed to be modular and reproducible, allowing for consistent environments across local development and TPU-based execution.

10.2.1 Requirements

Before starting, the project requires:

- Docker Desktop installed and running
- Google Cloud SDK installed and configured
- Python 3.11+ for local development
- Google Cloud account with billing enabled

- Service account with appropriate permissions
- Git for version control

10.2.2 Configuration

The configuration is managed through environment variables in a `.env` file. Key configuration parameters include: The service account requires the following permissions:

Table 10.1: Key Configuration Parameters

Parameter	**Description**
	PROJECT$_I D$
Google Cloud project identifier TPU$_R EGION$	Region for TPU deployment TPU$_Z ONE$
Zone within the region for TPU TPU$_N AME$	Name for TPU instance TPU$_T YPE$
TPU type (v2-8, v3-8, etc.) RUNTIME$_V ERSION$	TPU VM runtime version CONTAINER$_N AME$
Docker container name IMAGE$_N AME$	Docker image name with registry SERVICE$_A CCOUNT_J SON$
JSON key filename SERVICE$_A CCOUNT_E MAIL$	Service account email

- `roles/tpu.admin` - For creating and managing TPUs
- `roles/storage.admin` - For accessing GCS and Artifact Registry
- `roles/compute.admin` - For VM operations

10.2.3 TPU Zone Availability Check

The `check`$_z ones.shscriptautomatesfindingzoneswherethedesiredTPUtypeisavailable$: [1]$CheckZoneAvaila$

10.2.4 Docker Image Setup

`The setup`$_i mage.shscripthandlesbuildingandpushingtheDockerimage$: [1]$SetupDockerImageConfigureDoc$
$composePushimagetoGoogleContainerRegistry$

10.2.5 TPU VM Setup

`The setup`$_t pu.shscriptcreatestheTPUVMandconfigurestheDockercontainer$: [1]$SetupTPUVMCreateTPU$

10.3 Infrastructure Management Scripts

`The project includes specialised management scripts for working with the Docker container in the TPU environment.`

10.3.1 File Management Overview

Table 10.2: Management Script Functionality

Script	Purpose	Key Features
		mount.sh
Copies files to container	Directory structure preservation, full/partial mounting	Executes scripts or commands
	run.sh	
Python script execution, shell command execution, directory navigation	Keeps files in sync	Two-way synchronisation, dry run mode, selective updates
		scrap.sh
sync.sh		
Removes files from container	Selective/complete cleanup, directory preservation	

10.3.2 Mounting Files (`mount.sh`)

`The mount.sh script handles file mounting from local environment to the Docker container: [1] MountFilestarget, options options includes --all Prepare entire src directory structure options includes --dir Prepare directory for transfer Prepare single file for transfer Upload files to TPU VM Copy files from VM to Docker container with structure preservation Set appropriate permissions Key options include:`

- `--all: Mount the entire src directory structure`
- `--dir [directory]: Mount a specific directory and its contents`
- `--named-volumes: Use Docker named volumes instead of host directories`

10.3.3 Running Code (`run.sh`)

`The run.sh script executes code or commands in the Docker container:   [1] RunCodecommand options options includes --command Execute shell command in container Locate script in container script found Execute Python script with arguments Report error: script not found  Key options include:`

- `Regular mode: run.sh [script`$_{p}ath][arguments]-ExecutesaPythonscriptCommandmode:$ `run.sh --command "[command]" [directory]` $-Executesashellcommand$
- `--interactive, -i: Run command in interactive mode (with TTY)`

10.3.4 Synchronising Files (sync.sh)

The sync.sh script efficiently synchronises changed files: [1] SyncFilestarget, options Generate local file manifest Retrieve container file manifest Compare file timestamps to determine needed updates Create lists for files to update and files to delete options includes --dry-run Show what would be updated without making changes Process deletions for files that don't exist locally Transfer updated files to TPU VM Extract and copy to container with proper path preservation Key options include:

- --all: Sync all files in the src directory
- --dry-run: Show what would be updated without making changes
- --verbose: Show detailed information about file comparison

10.3.5 Cleaning Up Files (scrap.sh)

The scrap.sh script removes files from the Docker container: [1] CleanupFilestarget, options options includes --all Remove all files from mount directory options includes --dir Remove specified directory and contents Remove specified individual files Key options include:

- --all: Remove all files from the container mount directory
- --dir [directory]: Remove a specific directory and its contents
- File arguments: Remove specific files

10.3.6 Typical Workflow Patterns

Typical workflows combine these management scripts in specific patterns: [1] Mount entire src directory: ./infrastructure/mgt/mount.sh --all Run validation script: ./infrastructure/mgt/run.sh data/validate$_{d}ocker.py$ [1] Edit files locally Sync changes: ./infrastructure/mgt/sync.sh --all Run script with changes: ./infrastructure/mgt/ data/pipeline.py --model Transformer Check results: ./infrastructure/mgt/run.sh --command "ls -la" src/datasets/clean View outputs: ./infrastructure/mgt/run.sh --command "cat output.log" src Remove temporary files: ./infrastructure/mgt/scrap.sh --dir cache/prep Complete cleanup: ./infrastructure/mgt/scrap.sh --all

10.4 Data Preprocessing Features

The project includes robust data preprocessing capabilities optimised for TPU-based training.

10.4.1 Data Pipeline Overview

The data pipeline handles dataset downloading, preprocessing, and visualisation:
[1] DataPipelineoptions Load configuration from YAML options includes --download Download and prepare raw datasets options includes --view View raw or processed datasets Preprocess datasets options includes --optimize-for-tpu Apply TPU optimisation to processed datasets

10.4.2 Data Processing Options

The data pipeline supports comprehensive processing options:

Table 10.3: Data Processing Options

<table>
<tr><th>Option Category</th><th>Available Options</th></tr>
<tr><td></td><td>Model and Dataset Selection</td></tr>
<tr><td><code>--model [Transformer|static|all]</code>: Select model type</td><td><code>--dataset [gutenberg|emotion|all]</code>: Select dataset</td></tr>
<tr><td></td><td>Pipeline Control</td></tr>
<tr><td><code>--download</code>: Download and prepare raw datasets</td><td><code>--preprocess</code>: Preprocess datasets (default)</td></tr>
<tr><td><code>--view</code>: View datasets instead of processing</td><td><code>--force</code>: Force overwrite existing processed data</td></tr>
<tr><td><code>--disable-cache</code>: Disable caching of preprocessed data</td><td><code>--n-processes N</code>: Number of parallel processes to use</td></tr>
<tr><td></td><td>Resource Configuration</td></tr>
<tr><td><code>--config PATH</code>: Path to data configuration YAML file</td><td><code>--output-dir DIR</code>: Directory for processed outputs</td></tr>
<tr><td><code>--cache-dir DIR</code>: Directory for caching intermediate results</td><td><code>--raw-dir DIR</code>: Directory with raw datasets</td></tr>
<tr><td></td><td>Performance Options</td></tr>
<tr><td><code>--optimize-for-tpu</code>: Optimise preprocessing for TPU compatibility</td><td><code>--profile</code>: Enable performance profiling</td></tr>
<tr><td></td><td>Dataset Viewing</td></tr>
<tr><td><code>--dataset-type [raw|clean|auto]</code>: Type of datasets to view</td><td><code>--examples N</code>: Number of examples to show</td></tr>
<tr><td><code>--detailed</code>: Show detailed information about examples</td><td></td></tr>
</table>

10.4.3 Task Generation Capabilities

The framework includes comprehensive task generation capabilities:

10.4.4 TPU-Specific Optimisations

The preprocessing pipeline incorporates several TPU-specific optimisations:

- Padding to multiples of 8 for all tensor dimensions (critical for XLA)
- Fixed batch sizes that are multiples of 8 (preferably 128 per TPU core)

Table 10.4: Supported Task Generators

Task Type	**Description**	**Use Case**
		MLM
Masked Language Modeling	Standard BERT-style pre-training	Large span Masked Language Modeling
	LMLM	
Long-range dependency learning	Named Entity Recognition	Entity extraction with BIO tagging
NER		POS
Part-of-Speech tagging	Syntactic analysis	Next Sentence Prediction
	NSP	
Document coherence modeling	Discourse marker prediction	Rhetorical structure analysis
Discourse		Sentiment
Sentiment/emotion classification	Affective computing	Contrastive learning
	Contrastive	
Similarity encoding with validation		

- `Static shapes across all datasets to prevent XLA recompilations`
- `Memory-efficient implementations for maximum throughput`
- `Standardised array format outputs for direct TPU consumption`
- `BFloat16 precision for optimal numerical stability`
- `Length-based bucketing for efficient processing`

10.5 Resource Teardown

`The project provides scripts for clean resource termination to avoid ongoing charges.`

10.5.1 TPU VM Teardown

`The teardown`$_{t}pu.shscripthandlesTPUVMdeletion : [1]TeardownTPUVMValidateenvironmentvariablesC$

10.5.2 Docker Image Teardown

`The teardown`$_{i}mage.shscripthandlesDockerimagecleanup : [1]TeardownImageoptionsoptionsincludes - -lo$

`Interactive confirmation to prevent accidental deletions`

`Complete resource cleanup to avoid ongoing charges`

```
Proper GCP service disconnection

Local Docker image cleanup
```

10.6 Prototype in Lab4.ipynb

The `Lab4.ipynb` notebook in the `prototypes/` directory provides the foundational implementation of the model architectures and evaluation methodologies used in this project. This notebook contains comprehensive code for:

- Dataset selection and loading using Hugging Face's datasets library
- Data preprocessing including tokenization, cleaning, and exploratory visualization
- Implementation of four model architectures:
  - Model M1: Feed-forward network without attention
  - Model M2: Transformer variant with 1 attention layer
  - Model M3: Transformer variant with 2 attention layers
  - Model M4: Transformer variant with 3 attention layers
- 5-fold cross-validation training procedure
- Comprehensive evaluation metrics and visualizations
- Structural probing implementation for syntactic analysis
- Embedding visualization using UMAP and t-SNE

The notebook serves as both a proof-of-concept and development environment where the core algorithms were tested before being integrated into the full TAETPU framework. Its modular design allowed for iterative refinement of the model architectures and probing methodologies that form the foundation of the project's analytical approach.

10.7 Project Costs

The following table provides a breakdown of costs associated with this project: As shown in the cost breakdown, the primary expense is the Cloud TPU service, accounting for approximately 99

Table 10.5: Project Cost Breakdown

Service	**Service ID**	**Cost (£)**	**Discounts (£)**	**Promotions (£)**	**Unrounded (£)**	**Subtotal (£)**	**Change** heightCloud TPU
E000-3F24-B8AA	2283.71	0.00	0.00	2283.71	2283.71	New Compute Engine	6F81-5844-456A
11.59	0.00	0.00	11.59	11.59	New Artifact Registry	149C-F9EC-3994	7.60
0.00	0.00	7.60	7.60	New Cloud Storage	95FF-2EF5-5EA1	0.00	0.00
0.00	0.00	0.00	0Networking	E505-1604-58F8	1.08	-1.08	0.00
0.00	0.00	0height	**Subtotal**	2302.89	2302.89	height	**Tax**
0.00	0.00	height	**Total**	2302.89	2302.89	height	

Chapter 11

Original Project Brief

Problem Space Transformers have revolutionized development for Machine Learning (ML) and Large Language Models (LLMs). This Neural Network (NN) architecture allows ML models to encode words and their contextual (syntactic and semantic) meanings in text using a mechanism called Attention Layer or Attention Heads [5]. Attention Heads produce a set of multidimensional vectors or Word Embeddings (WEs) which are highly compatible with NN architectures, making the LLM training process more efficient.
Research has extensively focused on applying Attention Heads to maximise LLM and NN performance in Natural Language Processing (NLP) tasks to varying degrees of success[70]. It is no understatement to say that there is academic consensus on the benefits reaped by Attention and its applications in NLP. However, there is less agreement over how this relatively new technology works and what it does to data as it is being transformed across layers [17].

Project Goal This project seeks to understand how the Attention layer encodes contextual properties of words in Shallow Neural Networks (SNNs) with no more than 5 layers (including Attention). The aim is to quantify and graph text Information Flow (from words to WEs) using academic findings in Information Theory and Probing Techniques.

Project Scope

MUST

1. Perform research on probing techniques for NNs.
2. Construct a set of SNNs that learn from a small text corpus (Set A).
3. Construct the same set of SNNs, adding Attention layers to each model (Set B).
4. Train the created models on a small corpus of text.

SHOULD

5. Use findings to quantify information flow and contextual awareness of models in sets A, B .
6. Describe the workings of the Attention mechanism for models in Set B.

COULD

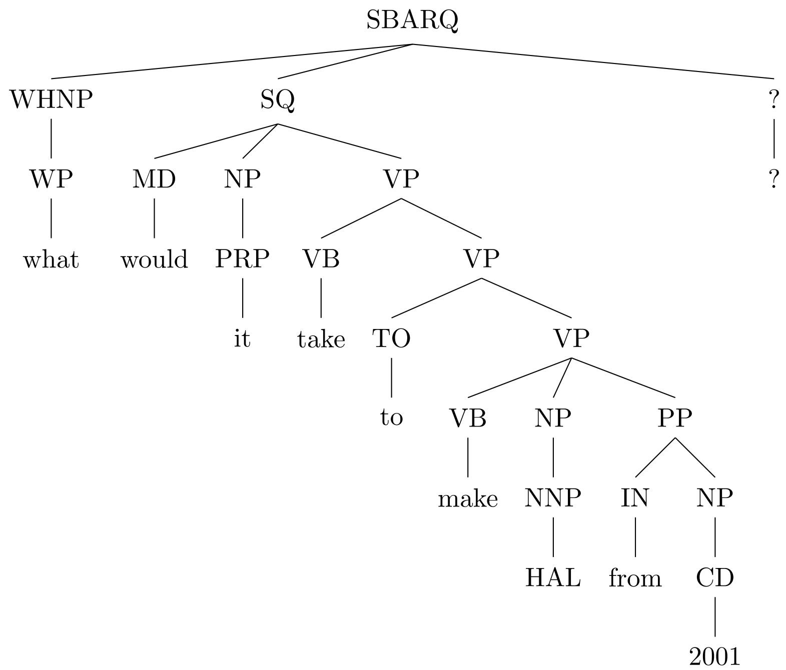
7. Extend findings to explore Attention heads in LLMs.

Chapter 12

Background Work

12.1 Abstract Syntax Trees (ASTs)

In this section, we provide a comprehensive explanation of each node used in the Context-Free Grammar (CFG) syntax tree for the sentence ”what would it take to make HAL from 2001?” as illustrated in Figure 2.2. Abstract Syntax Trees (ASTs) visually represent the hierarchical structure of the a sentence, illustrating how each linguistic component interacts to form a coherent question.



SBARQ Node

The **SBARQ** node, representing *Subordinate Clauses for Questions*, serves as the root of the syntax tree and encapsulates the entire question structure. Within this structure, the **WHNP** node, denoting a *Wh-Noun Phrase*, contains the wh-word that initiates the question, specifically the word "what". The **WP** node, or *Wh-Pronoun*, refers to "what" utilized within the WHNP.

SQ Node

The **SQ** node, standing for an *Inverted Yes/No Question Structure*, constitutes the main clause of the question that follows the wh-phrase, typically involving subject-auxiliary inversion. Within the SQ, the **MD** node represents a *Modal Auxiliary*, corresponding to the modal verb "would", which conveys a conditional or future-in-the-past aspect. The **NP** node, indicating a *Noun Phrase*, refers to the noun phrase "it" in the sentence. Associated with this, the **PRP** node, or *Personal Pronoun*, specifically identifies the pronoun "it" functioning as the subject of the verb.

VP Nodes

The **VP** node, denoting a *Verb Phrase*, encapsulates the main action or predicate of the sentence and is further divided into nested VPs to represent compound actions. Within the VP, the **VB** node stands for a *Base Verb* and represents the main verbs "take" and "make" present in the sentence. The **TO** node, an *Infinitive Marker*, refers to the word "to" used before the verb to form the infinitive. The **NNP** node, indicating a *Proper Noun*, refers to "HAL", serving as a proper noun that identifies a specific entity.

PP Node

The **IN** node, representing a *Preposition*, corresponds to the word "from", which establishes a relationship between "HAL" and "2001". The **CD** node, or *Cardinal Number*, refers to the numeral "2001" specifying a particular year. The **PP** node, denoting a *Prepositional Phrase*, represents the phrase "from 2001", providing additional information about "HAL". Finally, the **.** node, indicating a *Punctuation Mark*, corresponds to the question mark "?", which signifies the interrogative nature of the sentence.

12.2 Mathematical Foundations of MLP and FFNN Training

This appendix subsection builds upon the MLP framework and details the forward pass, training objectives (e.g., cross-entropy), backward pass, and the differences between a general Feed-Forward Neural Network (FFNN) and a classic Multilayer Perceptron (MLP) as detailed by Bishop's book and Rumelhart et al's implementation [4, 42].

Forward Pass of a Feed-Forward Neural Network

A Feed-Forward Neural Network (FFNN) maps inputs to outputs through a series of hidden layers, each transforming its inputs via a linear combination followed by a nonlinear activation. Consider an FFNN with L layers (excluding the input), where layer $\ell \in \{1, \ldots, L\}$ has M_ℓ units. Let the input be $\mathbf{x} \in \mathbb{R}^D$, with an augmented input $x_0 = 1$ to incorporate biases. The forward pass for layer ℓ is:

$$a_j^{(\ell)} = \sum_{i=0}^{M_{\ell-1}} w_{ji}^{(\ell)} z_i^{(\ell-1)}, \quad j = 1, \ldots, M_\ell, \tag{12.1}$$

where $z_i^{(\ell-1)}$ are the outputs of the $(\ell-1)$-th layer, and $w_{ji}^{(\ell)}$ are the weights connecting layer $(\ell-1)$ to layer ℓ. For convenience, define $z_0^{(\ell-1)} = 1$ to handle biases, and let:

$$z_j^{(\ell)} = h(a_j^{(\ell)}), \tag{12.2}$$

where $h(\cdot)$ is a differentiable nonlinear activation function, such as a logistic sigmoid, hyperbolic tangent, or ReLU. The network output $\mathbf{y}(\mathbf{x}, \mathbf{w})$ is given by the final layer's activations $z_j^{(L)}$.For an MLP, the architecture is a special case of an FFNN where layers are arranged sequentially, fully connected, and without additional complexities such as skip connections or alternative graph structures.

Training with a Cross-Entropy Loss

Training an FFNN involves choosing parameters $\mathbf{w}$ (weights and biases) that minimize a suitable loss function. Consider a supervised learning setting with N training samples $\{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$ where $\mathbf{t}_n$ is the target (e.g., a one-hot class vector for classification). For a classification problem with K classes, it is natural to use a softmax output layer. Let the final layer pre-activations be $\{a_k^{(L)}\}_{k=1}^K$, then:

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k^{(L)})}{\sum_{m=1}^K \exp(a_m^{(L)})}. \tag{12.3}$$

The cross-entropy loss for the entire dataset is:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w}). \tag{12.4}$$

This choice of output activation (softmax) and loss (cross-entropy) is statistically motivated by the likelihood principle for categorical distributions [4]. The cross-entropy loss encourages the network outputs to match the target class distributions.

Backward Pass and Backpropagation

The backward pass computes gradients $\frac{\partial E}{\partial w_{ji}^{(\ell)}}$ needed for optimization. Backpropagation systematically applies the chain rule starting from the output layer and moving backward through each layer. Define:

$$\delta_j^{(\ell)} = \frac{\partial E}{\partial a_j^{(\ell)}}, \tag{12.5}$$

which measures how sensitively the error E changes with respect to the pre-activation $a_j^{(\ell)}$ of unit j in layer ℓ. For output units (with a softmax output and cross-entropy error), we have:

$$\delta_k^{(L)} = y_k(\mathbf{x}, \mathbf{w}) - t_k. \tag{12.6}$$

For hidden units, we backpropagate δ values from the layer above:

$$\delta_j^{(\ell)} = h'(a_j^{(\ell)}) \sum_{k=1}^{M_{\ell+1}} w_{kj}^{(\ell+1)} \delta_k^{(\ell+1)}, \tag{12.7}$$

where $h'(a)$ is the derivative of the activation function evaluated at $a_j^{(\ell)}$. Having computed all $\delta_j^{(\ell)}$, we find the gradient w.r.t. a weight $w_{ji}^{(\ell)}$:

$$\frac{\partial E}{\partial w_{ji}^{(\ell)}} = \delta_j^{(\ell)} z_i^{(\ell-1)}. \tag{12.8}$$

This yields an $O(W)$ procedure for evaluating all gradients, where W is the number of parameters. The efficiency and local nature of backpropagation underlie its popularity for training neural networks.

12.3 Markov Models and MCMC

12.3.1 Markov Models

This section explains the early implementations of statistical Markov Models for language generation, as implemented by Shannon [38]. Let us consider a simple language consisting of the words: "start", "I", "love", "cats", "dogs", "end".We model this language using a Markov chain where each word represents a state. The transition probabilities between states are defined as follows:

$$A = \begin{array}{rc} & \begin{array}{ccccc} \text{I} & \text{love} & \text{cats} & \text{dogs} & \text{end} \end{array} \\ \begin{array}{c} \text{start} \\ \text{I} \\ \text{love} \\ \text{cats} \\ \text{dogs} \\ \text{end} \end{array} & \begin{bmatrix} 0.5 & 0.3 & 0.0 & 0.0 & 0.2 \\ 0.0 & 0.6 & 0.4 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \end{array}$$

Each entry a_{ij} represents the probability of transitioning from state i to state j. To generate a sentence, we begin in the "start" state. Then, the model automatically generates sentences by following the highest transition probabilities. For instance, let

$$\begin{aligned} \text{start} \rightarrow \text{I} &\quad \text{with probability } 0.5 \\ \text{I} \rightarrow \text{love} &\quad \text{with probability } 0.6 \\ \text{love} \rightarrow \text{cats} &\quad \text{with probability } 0.5 \\ \text{cats} \rightarrow \text{end} &\quad \text{with probability } 1.0. \end{aligned}$$

Unsurprisingly, these probabilities lead to the sentence "I love cats". Markov Chain Monte Carlo Methods The proposed MCMC method as per MacKay and Neal's works [43, 44] provides a framework for sampling from complex probability distributions, by quantifying uncertainty in predictions. Let us consider a probabilistic language model that assigns a probability $P(w_1, w_2, \ldots, w_T)$ to a sequence of words $\mathbf{w} = (w_1, w_2, \ldots, w_T)$. The goal is to predict the next word w_{T+1} given the previous words $\mathbf{w}$:

$$P(w_{T+1} \mid \mathbf{w}) = \frac{P(\mathbf{w}, w_{T+1})}{P(\mathbf{w})}. \tag{12.9}$$

To capture uncertainty in predictions, we adopt a Bayesian framework, treat language model parameters θ as random variables with a prior distribution $P(\theta)$. Given observed data $\mathcal{D} = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(N)}\}$, the posterior distribution of the parameters is given by Bayes' theorem:

$$P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta)P(\theta)}{P(\mathcal{D})} \tag{12.10}$$

We can use (18) to express predictive distribution for the next word w_{T+1} by integrating over the posterior distribution ofθ:

$$P(w_{T+1} \mid \mathbf{w}, \mathcal{D}) = \int P(w_{T+1} \mid \mathbf{w}, \theta)P(\theta \mid \mathcal{D})\, d\theta \tag{12.11}$$

Sadly, this integral is often intractable due to the high dimensionality of θ and the complexity of $P(\theta \mid \mathcal{D})$. This is where the Monte Carlo approximation of integrals comes in handy. MCMC methods approximate the integral by generating samples $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}\}$ from the posterior distribution $P(\theta \mid \mathcal{D})$. These samples can then be used to estimate the predictive distribution.

Sampling from the Posterior

One common MCMC algorithm is the Metropolis-Hastings (MH) algorithm, which constructs a Markov chain whose stationary distribution is the posterior $P(\theta \mid \mathcal{D})$. We can summarize this method as follows:

1. Initialize $\theta^{(0)}$ from an arbitrary distribution.
2. For $m = 1$ to M:
   (a) Propose a new state θ' from a proposal distribution $Q(\theta' \mid \theta^{(m-1)})$.
   (b) Compute the acceptance ratio:
   $$\alpha = \min\left(1, \frac{P(\mathcal{D} \mid \theta')P(\theta')Q(\theta^{(m-1)} \mid \theta')}{P(\mathcal{D} \mid \theta^{(m-1)})P(\theta^{(m-1)})Q(\theta' \mid \theta^{(m-1)})}\right)$$
   (c) Accept θ' with probability α. If accepted, set $\theta^{(m)} = \theta'$; otherwise, set $\theta^{(m)} = \theta^{(m-1)}$.

Using the newly obtained samples our predictive distribution can be approximated as:

$$P(w_{T+1} \mid \mathbf{w}, \mathcal{D}) \approx \frac{1}{M}\sum_{m=1}^{M} P(w_{T+1} \mid \mathbf{w}, \theta^{(m)}) \tag{12.12}$$

Estimating uncertainty in next word prediction

Let us set the hypothetical scenario of predicting the next word in the sequence "The cat sits on the [blank]"; we first assume a trigram model where θ represents the transition probabilities between word triplets. Given observed trigrams from a corpus, the posterior distribution of θ captures the uncertainty in the transition probabilities. Then, Using the Metropolis-Hastings algorithm, we generate samples $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}\}$ from $P(\theta \mid \mathcal{D})$. For each sampled $\theta^{(m)}$, we compute the probability distribution over possible next words:

$$P(w_{T+1} \mid \mathbf{w}, \theta^{(m)}) = \text{Trigram Probability based on } \theta^{(m)} \tag{12.13}$$

Lastly, we aggregate the predictions to express uncertainty, these probabilities reflect the uncertainty behind the the model's predictions:

$$P(w_{T+1} \mid \mathbf{w}, \mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^{M} P(w_{T+1} \mid \mathbf{w}, \theta^{(m)}). \tag{12.14}$$

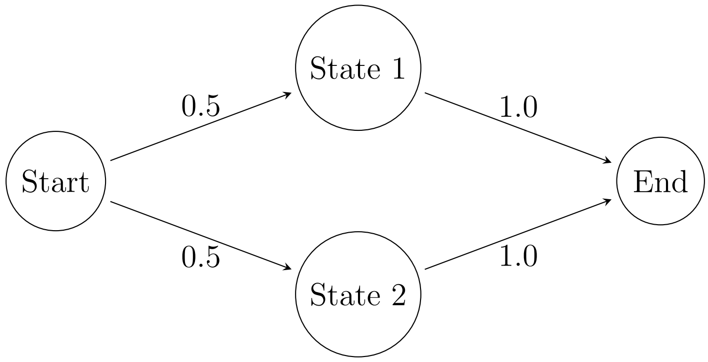


Figure 12.1: Example Markov Chain for Language Prediction

12.4 Neural Probabilistic Language Models

This section describes neural probabilistic language models that learn a joint probability function over word sequences as first introduced by [71]. In NLPMs, each word in the vocabulary V is associated with a vector in $\mathbb{R}^m$. The mapping $C : V \to \mathbb{R}^m$ yields a vector representation for each word. The parameter matrix C is of dimension $|V| \times m$. The joint probability of a word sequence $w_1, \ldots, w_T$ is factored into conditional probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-n+1}, \ldots, w_{t-1}) \tag{12.15}$$

The model order n determines the number of previous words conditioning the next word.

Neural Network Architecture The conditional probability $P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$ is expressed by a neural network function g. This network maps the distributed representations of context words into output probabilities for the next word:

$$f(i, w_{t-n+1}, \ldots, w_{t-1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$$

The input layer is formed by concatenating the feature vectors of the $n-1$ context words:

$$\mathbf{x} = (C(w_{t-n+1}), \ldots, C(w_{t-1})) \in \mathbb{R}^{(n-1)m}$$

Next, the network computes a hidden representation:

$$\mathbf{a} = \tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$$

The unnormalized log-probabilities $\mathbf{y}$ are then:

$$\mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{a}$$

The output probabilities are obtained via the softmax function:

$$P(w_t = i \mid \text{context}) = \frac{e^{y_i}}{\sum_{j=1}^{|V|} e^{y_j}}$$

Training Objective Model parameters $\theta = (C, W, U, b, d, H)$ are trained by maximizing the penalized log-likelihood of the training data:

$$L = \frac{1}{T} \sum_{t=1}^{T} \log f(w_t, w_{t-n+1}, \ldots, w_{t-1}; \theta) + R(\theta)$$

where $R(\theta) = \lambda(\|W\|^2 + \|U\|^2 + \|C\|^2)$. The parameters include $C \in \mathbb{R}^{|V|\times m}$, $W \in \mathbb{R}^{|V|\times(n-1)m}$, $U \in \mathbb{R}^{|V|\times h}$, $b \in \mathbb{R}^{|V|}$, $d \in \mathbb{R}^h$, and $H \in \mathbb{R}^{h\times(n-1)m}$. The total number of parameters is $|\theta| = |V|(1 + (n-1)m + h) + h(1 + (n-1)m)$.

Training via Stochastic Gradient Ascent Parameters are updated using stochastic gradient ascent:

$$\theta \leftarrow \theta + \epsilon \frac{\partial \log P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})}{\partial \theta}$$

12.4.1 GloVe and Word2ec

Word2Vec

Word2Vec is a group of related models that are used to produce word embeddings, which are dense vector representations of words capturing their semantic and syntactic meanings. Introduced by Mikolov et al. (2013), Word2Vec primarily consists of two architectures: Continuous Bag of Words (CBOW) and Skip-Gram.

Continuous Bag of Words (CBOW)

The CBOW model predicts the target word w_t given its surrounding context words $\{w_{t-m}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+m}\}$, where m is the window size. The objective is to maximize the average log probability of the target words given their contexts:

$$J_{\text{CBOW}} = \frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid \mathbf{C}_t), \tag{12.16}$$

where T is the total number of words in the corpus, and $\mathbf{C}_t$ represents the context vector derived from the surrounding words.

Probability Model

The probability $P(w_t \mid \mathbf{C}_t)$ is modeled using the softmax function:

$$P(w_t = w \mid \mathbf{C}_t) = \frac{\exp(\mathbf{v}_w^\top \mathbf{h}_t)}{\sum_{w'=1}^{|V|} \exp(\mathbf{v}_{w'}^\top \mathbf{h}_t)}, \tag{12.17}$$

where $\mathbf{v}_w \in \mathbb{R}^d$ is the output embedding vector for word w, $\mathbf{h}_t$ is the hidden layer representation, computed as the average of the input embedding vectors of the context words:$\mathbf{h}_t = \frac{1}{2m} \sum_{j=-m, j\neq 0}^{m} \mathbf{u}_{w_{t+j}}$, and $\mathbf{u}_w \in \mathbb{R}^d$ is the input embedding vector for word w.

Negative Sampling

To address the computational inefficiency of the softmax function with large vocabularies, Negative Sampling is employed. The objective function is reformulated to maximize the probability of the target word and minimize the probability of noise words:

$$J_{\text{NS}} = \log \sigma(\mathbf{v}_{w_t}^\top \mathbf{h}_t) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n(w)} \left[\log \sigma(-\mathbf{v}_{w_k}^\top \mathbf{h}_t) \right], \tag{12.18}$$

here $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, K is the number of negative samples, and $P_n(w)$ is the noise distribution, typically a unigram distribution raised to the $\frac{3}{4}$ power.

Skip-Gram

The Skip-Gram model, in contrast to CBOW, predicts the context words given a target word. The objective is to maximize the average log probability of the context words given the target word:

$$J_{\text{Skip-Gram}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{j=-m, j \neq 0}^{m} \log P(w_{t+j} \mid w_t). \tag{12.19}$$

Probability Model

Similar to CBOW, the probability $P(w_{t+j} \mid w_t)$ is modeled using the softmax function:

$$P(w_{t+j} = w \mid w_t) = \frac{\exp(\mathbf{v}_w^\top \mathbf{u}_{w_t})}{\sum_{w'=1}^{|V|} \exp(\mathbf{v}_{w'}^\top \mathbf{u}_{w_t})} \tag{12.20}$$

Negative Sampling

Negative Sampling is also applied in the Skip-Gram model to approximate the softmax:

$$J_{\text{NS}} = \sum_{j=-m, j \neq 0}^{m} \left[\log \sigma(\mathbf{v}_{w_{t+j}}^\top \mathbf{u}_{w_t}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n(w)} \left[\log \sigma(-\mathbf{v}_{w_k}^\top \mathbf{u}_{w_t}) \right] \right], \tag{12.21}$$

for $\mathbf{u}_w \in \mathbb{R}^d$, the input embedding vector for w, and $\mathbf{v}_w \in \mathbb{R}^d$, the corresponding output embedding of w.

GloVe (Global Vectors for Word Representation)

GloVe is a word embedding method that leverages global word-word co-occurrence statistics from a corpus [72]. Unlike Word2Vec, which is predictive, GloVe is a count-based model that combines the advantages of global matrix factorization and local context window methods.

Co-Occurrence Matrix

GloVe starts by constructing a word-word co-occurrence matrix X, where each entry X_{ij} represents the number of times word j appears in the context of word i. The context is defined within a certain window size around the target word:

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1|V|} \\ X_{21} & X_{22} & \cdots & X_{2|V|} \\ \vdots & \vdots & \ddots & \vdots \\ X_{|V|1} & X_{|V|2} & \cdots & X_{|V||V|} \end{bmatrix} \tag{12.22}$$

In this context, each word i is associated with two vectors: $\mathbf{w}_i \in \mathbb{R}^d$ the word vector, and $\tilde{\mathbf{w}}_i \in \mathbb{R}^d$, the context word vector. To facilitate training, each word and context word has a bias term, respectively: b_i, and $\tilde{b}_j$.

GloVe Objective Function

The core idea of GloVe is to find word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. The objective function is defined as:

$$J = \sum_{i,j=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2, \tag{12.23}$$

where $f(X_{ij})$ is a weighting function that assigns less weight to very frequent or very rare co-occurrences to balance the influence of different counts. The weighting function $f(x)$ is typically defined as:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise,} \end{cases} \tag{12.24}$$

where α is a hyperparameter (commonly set to 3/4) and $x_{\max}$ is a cutoff value to limit the influence of very frequent co-occurrences.

Training Procedure

GloVe optimizes the objective function J using stochastic gradient descent or other optimization algorithms. The gradients with respect to the parameters are computed as follows:

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{w}_i} &= \sum_{j=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right) \tilde{\mathbf{w}}_j, \\ \Rightarrow \frac{\partial J}{\partial \tilde{\mathbf{w}}_j} &= \sum_{i=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right) \mathbf{w}_i, \\ \Rightarrow \frac{\partial J}{\partial b_i} &= \sum_{j=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right), \\ \Rightarrow \frac{\partial J}{\partial \tilde{b}_j} &= \sum_{i=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right). \end{aligned} \tag{12.25}$$

Optimization Steps

The progress of optimization can be summarized as follows:

1. Initialize $\mathbf{w}_i, \tilde{\mathbf{w}}_j, b_i, \tilde{b}_j$ randomly.

2. Iterate for each non-zero entry X_{ij} in the co-occurrence matrix.

3. Compute the prediction error:

$$e_{ij} = \mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij}$$

4. Update the parameters using gradient descent:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \eta f(X_{ij}) e_{ij} \tilde{\mathbf{w}}_j$$

$$\tilde{\mathbf{w}}_j \leftarrow \tilde{\mathbf{w}}_j - \eta f(X_{ij}) e_{ij} \mathbf{w}_i$$

$$b_i \leftarrow b_i - \eta f(X_{ij}) e_{ij}$$

$$\tilde{b}_j \leftarrow \tilde{b}_j - \eta f(X_{ij}) e_{ij}$$

5. Iterate over the co-occurrence matrix until convergence.