**ARTIFICIAL INTELLIGENCE FINAL ASSIGNMENT**

**REPORT: SCIENTIFIC ABSTRACT CLASSIFICATION**

Andifallih Noor Malela – 26002304448

31889: Artificial Intelligence (G1)

**INTRODUCTION**

Text classification is a common task in the field of machine learning (ML) and natural language processing (NLP), a field of ML focused on understanding language. It usually consists of preprocessing input dataset with techniques such as stemming, stop-word removal, and/or tokenization, followed by applying machine learning or deep learning algorithms such as random forests, support vector machines (SVM), or transformer-based models like BERT (Bidirectional Encoder Representations from Transformers).

In this report, the author attempted create a classifier that’s able to categorize scientific paper abstracts into one of three predefined scientific fields using BERT-base model. The three scientific chosen are political science, sociology, and psychology.

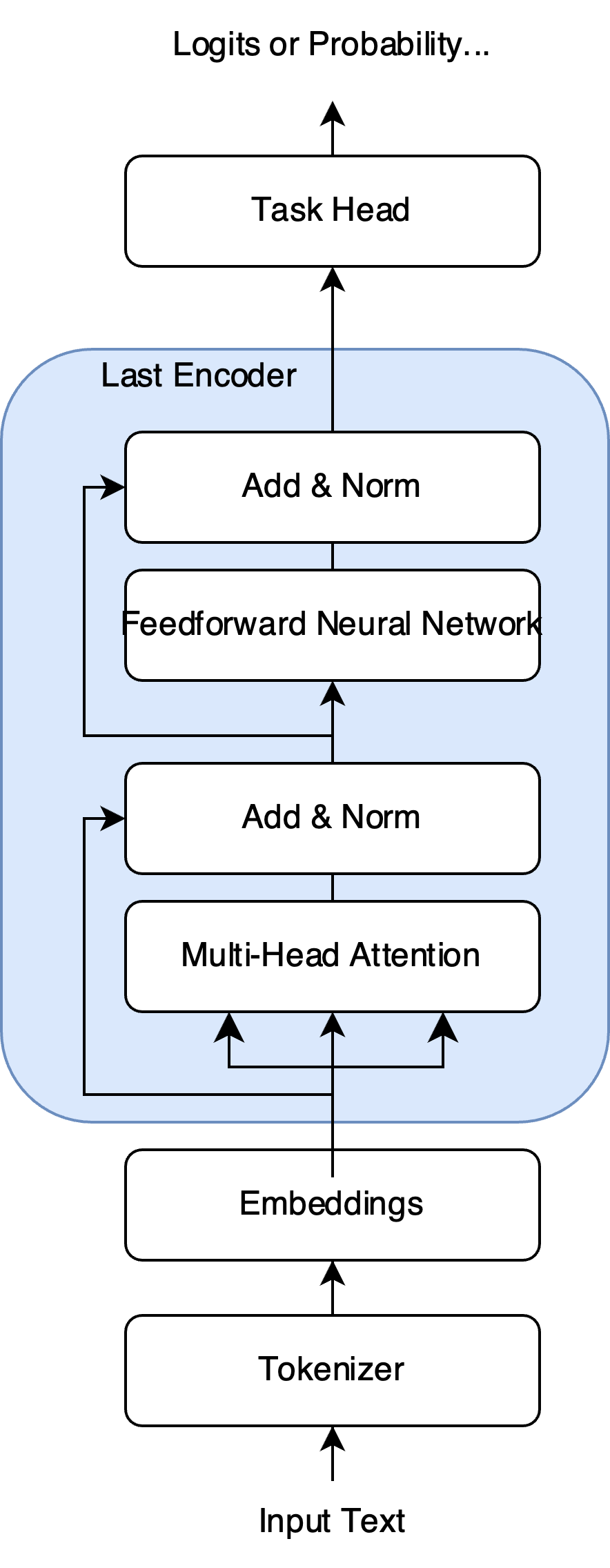
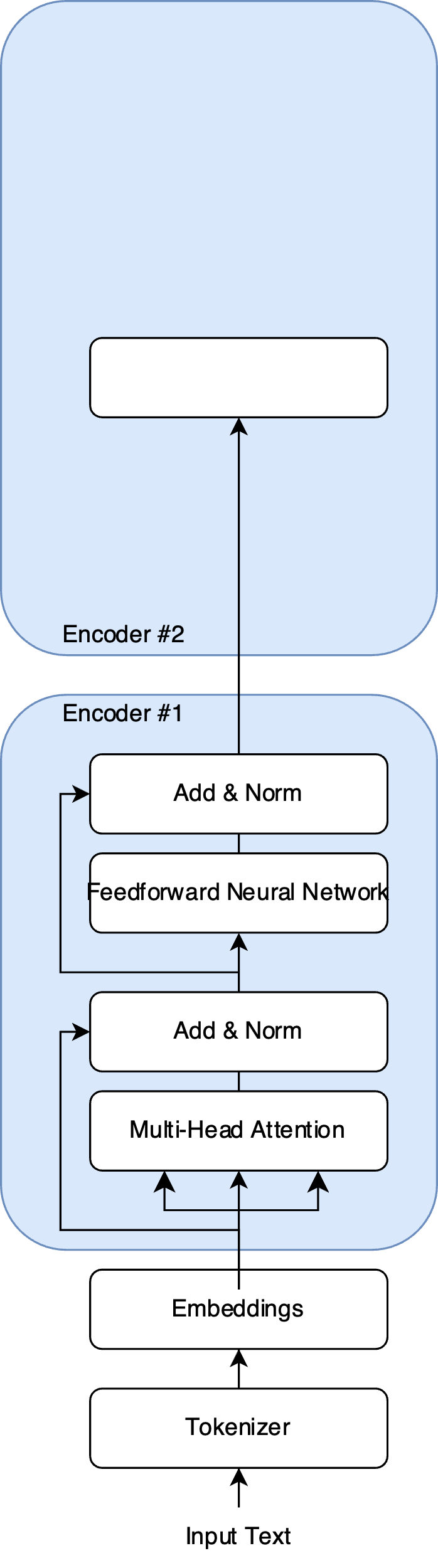
**BACKGROUND OF BERT**

BERT (Bidirectional Encoder Representations from Transformers) is a language model pre-trained to understand the English language and can be fine-tuned for a variety of tasks, including text classification.

* *Bidirectional*: BERT simultaneously processes texts from both left and right directions of a word to consider the full “bidirectional” context of a word to its surrounding texts.
* *Encoder*: The encoder component in a Transformer architecture which BERT relies on.
* *Representations*: BERT generates representations that capture contextual relationships and meanings of words.
* *Transformers*: The deep learning architecture that consists of encoders and decoders. However, BERT is only uses the encoder portion of a Transformer within its architecture.

From a high-level, the pipeline of BERT consists of 4 components: Tokenizer, Embedding, Encoder, and Task head.

1. **Tokenizer**



Preprocesses English words into format BERT can understand. It breaks input sentences into sub-words “tokens" which are mapped to a unique ID based on the tokenizer’s vocabulary. It also adds special tokens [CLS] for classification at the beginning and [SEP] for sequence separator. BERT-base’s tokenizer, WordPiece, handles tokenization without requiring much preprocessing such as lemmatization, stemming, stop-word removal.

* Text: “I am training NLP”
* Tokens: “[CLS], i, am, train, ##ing, nlp, [SEP]” – *lowercases if the tokenizer used is uncased*
* Token IDs: “[101, 1045, 2572, 4735, 2075, 17953, 102]”

Tokenizer also truncates sequences that are too long and adds padding [PAD] token for sequences that are too short, making all input equal in to specified length (e.g., 512), with real tokens with marked with 1 and padding tokens with 0 (attention masking). Only real tokens will be focused by the model in computation.

1. **Embedding**

Embeddings are dense vectors (vectors of mostly non-zero, real-valued numbers as values) with the goal to represent higher dimensions of textual information from the tokenized input data. Each token is converted to embeddings consisting of three types: token, segment, position.

**Token Embedding**

The token IDs access a dense vector specified in an embedding matrix: an array of dense vectors where each vector represents a unique token in the vocabulary. WordPiece has vocabulary size of 30000 and embedding size of 768. So, the token embedding matrix size is 30000768 with each token embedding being a 768-dimensional dense vector.

For example, [CLS] with ID [102] accesses vector in row 102 of the matrix, which becomes its token embedding E[CLS].

**Segment Embedding**

Segment embeddings differentiate tokens in multiple-input sequences. Tokens in the same segment share the same segment embedding. There are two segment embeddings in BERT, EA and EB from a 2768 segment embedding matrix.

* Tokens: “[CLS], how, are, you, ?, [SEP], good, [sep]”
* Segment embedding: “EA, EA, EA, EA, EA, EA, EB, EB”

This however is only for multiple-input tasks such as question-answering and next-sentence prediction such as during pre-training BERT (notice the middle [SEP] added to indicate differing segments). Otherwise, for single-input tasks such as text classification, all tokens belong to one segment.

* Text: “I am training NLP. It is fun”
* Tokens: “[CLS], i, am, train, ##ing, nlp, ., it, is, fun, [SEP]”
* Segment embedding: “EA, EA, EA, EA, EA, EA, EA, EA, EA, EA, EA”

**Position Embedding**

Used to understand token position in the sequence. Each token is assigned an embedding from a position embedding matrix according to its index in a sequence. The maximum sequence length in BERT-base is 512. Therefore, the max size of the position embedding matrix is 512768.

* Tokens: “[CLS], i, am, train, ##ing, nlp, [SEP]”
* Position embedding: “E0, E1, E2, E3, E4, E5, E6, E7”

When training the model, values in all the embedding matrices and corresponding embeddings are updated though backpropagation as the model understand contextual information within input data.

Finally, an input embedding for each token is computed:

The full sequence of input embeddings is a matrix with size . The is sequence length and is embedding size. This matrix is then processed through the encoder layers.

1. **Encoder**

Encoder is responsible in making contextual representations for every token in the input sequence. Each encoder consists of two major components: a multi-head attention mechanism and a feedforward network (FFN) each followed by an add & norm module.

**Multi-Head Attention**

Self-attention mechanism determines relative importance of each token to every other token in the sequence, allowing entire context surrounding each word be fully considered giving a bidirectionality aspect to the model. Because, in natural language, word meanings depend on other words.

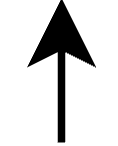
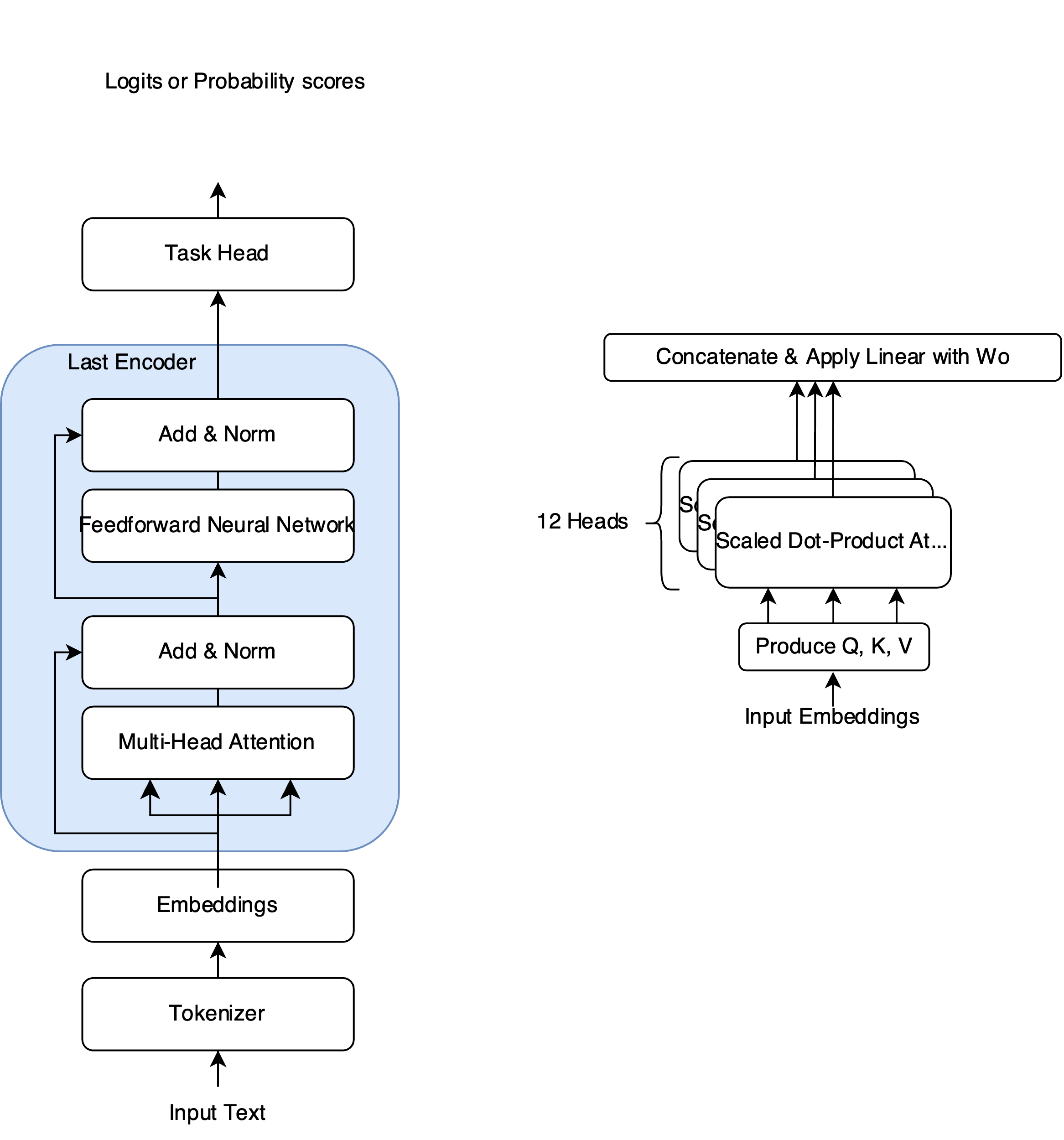
A single attention mechanism might only focus on one type of relationship between tokens. So, multi-head attention aims to capture multiple different types of relationships and features by splitting the self-attention mechanisms to multiple attention heads (12 heads in BERT-base) and each head running parallelly. Here’s how it works:

Queries , keys and values matrices are created from input embedding :

are weight matrices with values learned during training.

For *each head*, scaled-dot product attention calculates how much each token attends to all other token in the sequence:

This is computed independently for each head. Then the multi-head output for all number of heads is calculated:



is projection matrix learned during training.

The concatenation combines different types of relationship representations from each head, while the acts like an editor which learns how to combine them meaningfully as the model is trained for a certain task.

**Add & Norm**

There is add & norm module after each multi-head attention and FFN layers. Here are the functions:

* Add: Preserves original information in input data, directly combining them to the new learned transformation (after the multi-head attention and FFN layers).
* Norm: Normalizes output of transformations, ensuring values remain stable, preventing exploding or vanishing gradients during training.

After multi-head attention:

After FFN:

**FFN (Feedforward Network)**

Multi-head attention gives linear context to each token representations based only off relationships captured by attention mechanism. FFN introduces non-linearity through activation functions allowing the model to learn even more complex patterns and refine token representations.

There are two layers in each FFN module. First layer expands dimensionality of embeddings enabling it to learn more complex relations and patterns. The second layer reduces the dimensionality back to its original size retaining its learned information. Here’s the formula:

is a normalized multi-head attention output matrix and is activation function.

Then it goes through add & norm module again before going to another encoder layer or a task head.

BERT-base contains 12 encoder layers. Each layer refines token representation capturing more complex relationships between tokens.

1. **Task Head**

BERT-base contains 12 encoder layers. Each layer refines token representation capturing more complex relationships between tokens. After the last encoder, the output is a matrix consisting of contextualized embeddings of tokens. This is then processed to a task head is positioned after the last encoder that are specific to the tasks.

BERT-base was pre-trained on Masked language modeling (MLM) task and next sentence prediction (NSP) task.

* MLM: the task head is an un-embedding layer. During this task, random tokens will be replaced with [MASK] token. The training objective is to predict these masked tokens by projecting embeddings back to vocabulary and generating correct token probabilities.
* NSP: the task head is a binary classification layer. Given two sequences of input data, the model tries to predict if the second sequence logically follows the first.

Text classification task to three distinct classes (the fine-tuning goal of this report) is like the NSP task, however it uses a single sequenced input data. In more details, it works by mapping the [CLS] token embedding, which aggregates information about the entire input sequence, through a fully connected layer (networks where each input neurons is connected to every output neurons with learnable weights) and a softmax function (converts raw output scores aka logits to probability distribution over the classes) or argmax function (selects largest possible values in the classes).

**METHODOLOGY**

1. **Dataset Acquisition, Cleaning, Preprocessing**

The abstracts were obtained by scraping scientific journals specific to a given field using CrossRef API,with the script written in Python. CrossRef API allows access to metadata of scholarly articles, including abstracts. The script makes a request to CrossRef API, filtering the results based on a specified journal that retrieves an abstract. It then saves them to a CSV file each with a label corresponding to the field of the specified journal (e.g., "sociology"). The scraper performs this for each field. Here is the list of journals used:

* Political Science: Annual Review of Political Science, British Journal of Political Science.
* Psychology: Annual Review of Clinical Psychology, Psychological Science.
* Sociology: Sociology, Annual Review of Sociology, American Journal of Cultural Sociology.

After scraping, to make sure that the datasets only contain texts relevant to the abstract cleaning involved removing:

* Duplicates.
* Unicode characters.
* Empty entries like “Not available” and blank strings.
* HTML artifacts.
* Whitespaces and line breaks.
* Text artifacts such as “Abstract:” and “Conclusion:” to prevent interference with training – certain journals returned these text in the abstracts while others did not, the model might learn to associate on these terms instead of focusing on the abstract content.

One thing to note was that the author performed data acquisition before deciding on which models to use for text classification. Therefore, it included steps that are unnecessary when using BERT tokenizer such as removing Unicode characters and whitespaces.

The resulting dataset is a collection of 900 total abstracts from three scientific fields with each field having 300 abstracts.

1. **Learning Tools**

PyTorch was used as the foundational framework in training and fine-tuning BERT.

K-fold cross-validation technique was employed used during training to evaluate the model's performance on different subsets of the data. There are k number of folds, and the dataset was divided to k number of splits. The model trains on k-1 splits with the remaining split used for validation, and the splits were rotated for each fold. This ensured that every datapoint is used for both training and validation. The folds were stratified using library to ensure that proportion of the class labels in each split dataset is consistent with their proportion in the original dataset. 1/3rd of the data being psychology labels, 1/3rd being sociology labels, and 1/3rd being political science labels. To do these tasks, StratifiedKFold library was used.

ReduceLROnPlateau was used to adjust the learning rated during training based on the performance of validation loss metric. If the loss does not improve after a certain number of epochs, learning rate is reduced by a certain factor.

Adam (Adaptive Moment Estimation) was used to optimize the model’s parameters during training by applying the current learning rate. It uses gradient decent with momentum and RMSP (Root Mean Square Propogation) algorithms. Gradient descent with momentum removes sudden changes in parameter values, smoothing it and fastens training, while RMSP adapts the learning rate for each parameter based on previous gradients.

**EXPERIMENTATION**

1. not having an early stopper for 3, 4, 5 epochs vs early stopper

initially i did not use an early stopper for simplicity but found that sociology often overfits while other fields are still learning

i started with 5 epochs with recommendation from GPT, then noticed that loss is still going down so i went to 6, then to 4, then to 3

2. fixed learning rate vs learning rate scheduler

3. cased vs uncased BERT model

**RESULTS**

**CONCLUSIONS**