TRANSFORMER-BASED REGRESSION MODELS FOR ASSESSING READING PASSAGE COMPLEXITY: A DEEP LEARNING APPROACH IN NATURAL LANGUAGE PROCESSING

Harmanpreet Sidhu and Amr Abdel-Dayem

Bharti School of Engineering and Computer Science, Laurentian University, Sudbury, Ontario, Canada

ABSTRACT

Natural Language Processing (NLP) is a vital area in deep learning, widely applied in tasks like text classification, virtual assistants, speech recognition, and autocorrect features in digital devices. It allows machines to understand and generate human language, enhancing user interactions with software. This paper presents a deep learning model using the Transformer architecture for a regression task to predict the complexity of reading passages based on text excerpts. By leveraging the Transformer's capability to identify complex patterns in text, the model achieves a relative error rate of about 10%. The paper also examines how different architectural choices influence model performance, focusing on one-hot encoding and embeddings. While one-hot encoding provides a simple text representation, embeddings offer a richer, more nuanced understanding of word relationships. The findings highlight the significance of model design and data representation in optimizing NLP tasks, providing insights for future advancements in the field.

KEYWORDS

Natural language processing, Transformer models, Regression models, word embeddings.

1. INTRODUCTION

Machine learning, a cornerstone of Artificial Intelligence, diverges from traditional approaches by relinquishing explicit instructions to computers in favour of providing examples for problemsolving. In this paradigm, the machine autonomously learns rules from the provided data. Within machine learning, deep learning has emerged as a prominent sub-field, characterized by models with numerous layers and their capacity to derive hierarchical representations from data. This multiplicity of layers, aptly named 'deep learning,' obviates the need for meticulous feature engineering, as the models proficiently extract relevant features during the learning process.

In comparison to its machine learning counterparts, deep learning reduces the necessity for human intervention. Machine learning demands structured data, necessitating human experts to discern the most pertinent features. Conversely, deep learning models excel in automatically discerning and extracting essential features, diminishing the reliance on human-guided structuring. Recent years have witnessed the pervasive success of deep learning across diverse tasks, including image classification, natural language processing, digital assistants, and autonomous driving.

Natural Language Processing (NLP) represents a pivotal domain aspiring to attain human-level comprehension of spoken or written language. The inherent challenges of NLP lie in the dynamic and ambiguous nature of language rules. Unlike earlier endeavors that aimed at deciphering the

DOI:10.5121/ijaia.2024.15504

intricacies of language, modern NLP focuses on efficiently processing language to yield practical outcomes, such as text classification, content filtering, translation, and summarization. Deep learning has garnered considerable attention in the realm of text manipulation, owing to its superior performance and reduced dependence on intricate feature engineering.

Within the domain of NLP, the Transformer architecture has emerged as the preeminent building block. This architectural paradigm facilitates the efficient processing of sequential data by acknowledging that not all components of a text warrant equal scrutiny. Section 4 provides more details on the Transformer architecture, unraveling its principles and contributions to the realm of NLP.

Classifying reading materials has wide-ranging applications, from enhancing educational content to improving recommendation systems. This can boost literacy by ensuring individuals are presented with reading material suited to their age and comprehension level, fostering a more adaptive learning environment. Additionally, the model greatly reduces the time and effort needed for manual classification, streamlining the process and increasing efficiency.

2. RELATED WORK

Recent years have witnessed a surge of interest and extensive research in Natural Language Processing (NLP) within the broader landscape of machine learning. The ubiquitous application of data prediction in various aspects of daily life has propelled the prevalence of regression models. Within the realm of deep learning models, the optimizer plays a pivotal role in determining the weight adjustments at each step. One optimizer that has garnered considerable attention is the RMSProp optimizer, renowned for its efficiency in both classification and regression tasks. A study conducted by Kurbiel and Khaleghian [1] explores the efficacy and accuracy of the RMSProp optimizer, also detailing how to use it to train multi-layer neural networks. Noteworthy findings include the utility of neural networks with layers featuring distance measures and Gaussian activation functions to counteract the vanishing gradient problem associated with dot-product-based activation functions. The current paper adopts RMSProp as the optimizer, with Section 4.6 delving into the rationale behind this choice.

Model regularization, a critical aspect preventing overfitting to training data, has also captivated researchers. Krueger et al. introduced the Zoneout technique, a selective application of dropout to specific features within a network [2]. This method introduces controlled randomness by preserving some unit values at each step, enhancing the model's generalization capacity. The optimizer employed in this paper combines zoneout and dropout techniques.

Within the specialized domain of NLP, research endeavors have focused on developing models capable of efficiently capturing textual representations in vector form [3], leveraging one-hot encoding for specific tasks [4], and scrutinizing word embeddings generated for deep learning [5]. These techniques find application in the current paper for the numerical representation of textual data, rendering it accessible to deep learning models.

3. PROBLEM STATEMENT

In this paper, a deep learning model was developed to evaluate the complexity of reading passages. The model was trained using samples that included an excerpt from each passage along with its corresponding complexity level. This approach enabled the model to learn and accurately predict the complexity of previously unseen passages.

This constitutes a regression task conducted through supervised learning. Unlike classification problems, regression tasks involve predicting a continuous value instead of a discrete label. The specific task undertaken here exemplifies scalar regression, wherein the model aims to predict a International Journal of Artificial Intelligence and Applications (IJAIA), Vol.15, No.5, September 2024 single continuous value. A function is derived by the machine, taking in inputs and generating a single output value. This function aids in predicting the value of a variable (referred to as the target or dependent variable) based on one or more independent variables. Supervised learning, utilized for labeled datasets, involves the machine classifying or predicting outputs based on these labels. Algorithms for supervised learning establish mappings between input data instances and their corresponding output values.

The inspiration for the deep learning model was drawn from a competition hosted on Kaggle [6], organized by CommonLit, a non-profit education technology organization. The objective was to formulate a model capable of rating the complexity of reading passages for classroom use.

This model holds potential benefits for educators, facilitating the selection of appropriate reading material based on student proficiency. Ensuring students receive literary material suitable for their age is crucial for developing the essential skill of reading. Additionally, it can serve as a valuable tool in curriculum planning, allowing educational providers to ensure a gradual increase in reading complexity over time,as well as adapting the educational content according to the individual learning levels. Another potential application of this model lies in a recommendation system, suggesting reading material to individuals based on their current language proficiency.

4. MODEL BUILDING

4.1. Introduction

The objective of this deep learning model is to forecast the reading level of a text passage, constituting a regression task executed through supervised learning. The resultant model assimilates the relationships between input features and their respective targets, empowering it to predict the complexity of any given reading passage.

4.2. Experimental Setup

TensorFlow, a Python-based open-source machine learning platform developed by Google, plays a pivotal role in deep learning by facilitating the automatic computation of gradients for differentiable expressions [7]. Its notable feature of easy distribution across multiple machines enhances scalability and performance in diverse applications.

Built on top of TensorFlow, Keras serves as a high-level deep learning API, streamlining the process of model development. Offering a variety of workflows, it allows the definition and training of deep learning models. Keras combines pre-built functions for common tasks with the flexibility to define custom functions or configure existing ones, providing a versatile environment for model construction [8].

In the development process of the model, Keras was utilized as the primary workspace, benefiting from its high-level abstractions and TensorFlow's underlying capabilities. This combination allowed for streamlined development, enabling quick experimentation and iteration to achieve optimal model performance.

4.3. Dataset

The data utilized for this model was sourced from Kaggle, an online platform facilitating data scientists in dataset publication, collaboration, and participation in competitive problem-solving. The model's objective aligns with the goal of a Kaggle competition [6].

4.3.1. Data Overview

The dataset comprises several columns:
id: A unique identifier for each entry.
url_legal: Source URL.
license: Literary source license.
Excerpt: The text passage whose complexity the model aims to predict (string datatype).
Target: The reading ease of the passage, the target variable (floating-point values).

Columns like id, url_legal, and license, while possibly included for copyright and legal purposes, were dropped as they lack relevance to the model's deep learning task.

4.3.2. Model Objective

The model's task is to predict the reading ease of passages, entirely dependent on the excerpt.
An example entry from the excerpt column and its corresponding target is depicted in Figure 1. Since deep learning models process tensors, vectorization of data in this column was imperative.

The general steps for vectorization of text data are as follows:

- Text Standardization: Standardizing text is a fundamental data manipulation step that encompasses actions like removing punctuation and converting data to lowercase. This is essential as it avoids confusion caused by punctuation, and the model would otherwise distinguish between the same word in different cases.
- Tokenization: Tokenization involves splitting the text into units suitable for vectorization. While there are various approaches, the commonly employed method is word-level tokenization, where tokens are substrings separated by spaces.
- Indexing: Following tokenization, the text undergoes conversion into a numerical representation. Indexing, in this context, entails creating an index encompassing all words in the data and assigning a unique integer to each word.
- Encoding: The final step involves converting integers from the previous indexing phase into vectors. Two widely utilized techniques for encoding are one-hot encoding and embeddings.

One beautiful misummer night in 18â€" a large, heavily laden steamer was making her way swiftly up the Pacific coast, in the direction of San Francisco. She was opposite the California shore, only a day's sail distant from the City of the Golden Gate, and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day was to intervene ere they could expect to set their feet upon solid land. She was one of those magnificent steamers that ply regularly between Panama and California. She had rather more than her full cargo of freight and passengers; but, among the hundreds of the latter, we have to do with but three. On this moonlight night, there were gathered by themselves these three personages, consisting of Tim O'Rooney, Elwood Brandon and Howard Lawrence. The first was a burly, good-natured Irishman, and the two latter were cousins, their ages differing by less than a month, and both being in their sixteenth year.	-1.48389

Figure 1. Example entry from the excerpt column and its corresponding target. A neural network cannot process the data in form as provided in the excerpt column.

4.3.2.1. One-Hot Encoding

One-hot encoding is a technique where each word in a dataset is transformed into a vector with all zeroes, except for a 1 in the position corresponding to the word's index. For instance, in a dataset with only the words 'Cat,' 'Dog,' and 'Horse,' the one-hot encoded vectors will each have three elements. If indices are assigned based on the word order, 'Cat' will be [1, 0, 0], 'Dog' will be [0, 1, 0], and 'Horse' will be [0, 0, 1].

The dimensionality of one-hot encoded vectors equals the number of words in the dataset, resulting in generally high-dimensional and sparse vectors. These vectors are orthogonal, meaning they are independent of each other. One-hot encoding, representing words solely as indices in a vocabulary, lacks the concept of similarity between words. A variant, multi-hot encoding, allows encoding multiple words in a single vector, similar to one-hot encoding, where the encoded vector includes all zeroes except for indices indicating specific words.

4.3.2.2. Embedding

Word embedding involves creating vectors that represent a given text in a structured geometric space. In this space, words with related meanings or semantic similarities are positioned closer to each other, while words with distinct meanings are situated farther apart. Embedded words retain their individual identities while sharing attributes with similar words, fostering a semantic clustering effect based on categories.

Figure 2. Representations obtained through one-hot encoding: high-dimensional and sparse.

Taken from [9].

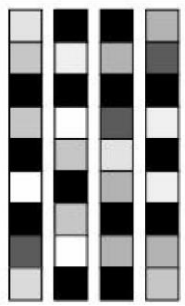

Figure 3. Representations obtained through embeddings: low-dimensional and dense. Taken from [9].

These vectors crafted through embeddings are characterized by being low-dimensional and dense. Unlike one-hot encoded vectors, machine learning algorithms autonomously learn word embeddings without explicit programming by developers. Recent attention from researchers has propelled word embeddings into various applications, including text classification, sentiment analysis, and knowledge mining. To illustrate the contrast between representations, Figure 2 depicts those from one-hot encoding, while Figure 3 showcases representations from word embeddings.

While one-hot encoding proves valuable in many deep learning algorithms, its use was impractical for this paper. The relatedness of words in a passage necessitates a holistic understanding, making individual word examination insufficient for determining the complexity of the entire excerpt.

4.3.3. Generating Embeddings

Word embeddings are tailored to the specific task at hand, varying across different applications. They originate from a corpus and offer versatility across multiple use cases. Two methods are commonly employed for their generation:

- Task-Specific Development: In this approach, word embeddings are crafted concurrently with the task they are intended for. Similar to how a machine learns the weights of a network, this method enables the machine to learn task-specific word embeddings.
- Pretrained Embeddings: Alternatively, researchers often opt for pretrained word embeddings developed for a different machine learning task. This practice involves using embeddings generated from a large corpus, bypassing the need to create embeddings from scratch for a given dataset. This preference arises due to the computational demands, preprocessing requirements, and training time associated with developing custom embeddings.

4.4. Model Architecture

A critical decision in natural language processing tasks involves determining how to encode the order of words in a sentence, shaping the foundation of the model's architecture. Two primary approaches exist: one treating the text as an unordered collection of words and the other processing words sequentially, akin to time-series data, often utilizing Recurrent Neural Networks (RNNs).

For the current task, where understanding the sentence structure is paramount, a hybrid approach is adopted. The Transformer architecture is employed, striking a balance by incorporating information about the position of words in their representations. This approach outperforms simply using RNNs, enabling the model to simultaneously consider various parts of the sentence, enhancing overall comprehension.

4.4.1. Transformer Architecture

The Transformer architecture, rooted in neural attention mechanisms, has emerged as the cornerstone for NLP models. Vaswani et al. introduced the concept of "attention" in their seminal paper "Attention is All You Need" [10]. This idea suggests that certain features of input data merit more attention from the model, while others can be selectively ignored.

Widely utilized for sequence-to-sequence learning, the Transformer architecture comprises two key components: an encoder and a decoder. The encoder processes input data, transforming it into an intermediate representation, which the decoder leverages to predict the output.

4.4.2. Sentence Transformer

A specialized type of NLP model, Sentence Transformers leverage the Transformer architecture to generate embeddings that encapsulate the semantic meaning of textual data. These embeddings are highly versatile and can be applied to a variety of tasks, including text classification, sentiment analysis, and paraphrase detection [11]. Notable for their adaptability to diverse datasets, Sentence Transformer models are used in this paper, as visualized in Figure 4, which illustrates the transformed embeddings derived from the excerpt column.

```
0    [-0.022620749, 0.14137924, 0.030773994, 0.0443... -0.340259
1    [-0.097603306, 0.023281628, 0.06266469, 0.0255... -0.315372
2    [-0.07508897, 0.03696774, 0.07279821, -0.09623... -0.580118
3    [0.030975923, 0.11476946, 0.023467604, 0.02791... -1.054013
4    [0.069664896, 0.062562086, 0.059518028, 0.0879...  0.247197
```

Figure 4. Excerpt column after conversion to Embeddings. This data can now be processed by the model as it is provided in the form of floating-point tensors.

The process begins with converting the input text into individual tokens. Each token is then mapped to a pre-trained embedding vector available in the model’s vocabulary, resulting in dense vector representations. Unlike traditional models, Transformer models create contextaware embeddings, meaning each word's embedding captures both its meaning and its relationship to other words in the sentence. This is achieved by analyzing the entire input sequence simultaneously, rather than processing words in isolation.

4.5. Model Building

4.5.1. Layer Structure

The number of layers in a model is intricately linked to the complexity of the representations it aims to learn. The dimensionality of the input data determines the number of units in each layer. While a larger model with more layers and units can capture intricate representations, it might also learn extraneous patterns, making it susceptible to overfitting. Conversely, a smaller model

may struggle to grasp robust representations. For this task, the initial model comprised 5 Dense layers, with the first 4 layers having 64 units each. Dense layers, being fully connected, enable each neuron to receive input from all neurons in the preceding layer. The first 4 layers employed the Gelu (Gaussian Error Linear Unit) activation function, detailed in the subsequent section. The last layer, crucial for scalar regression, featured a linear configuration without any activation function to allow predictions across a broad range.

4.5.2. Activation Function - Gaussian Error Linear Unit (Gelu)

Selecting activation functions is pivotal in model development. Optionally, stochastic regularizers like dropout or noise addition can enhance performance. Gelu, denoting Gaussian Error Linear Unit, amalgamates two stochastic regularizers—dropout and zoneout—with the traditional relu activation function. Dropout randomly omits some output features during training, preventing overfitting. Zoneout, akin to dropout, injects noise during training by retaining some unit values, addressing the vanishing gradient problem. Gelu multiplicatively combines these techniques by introducing noise based on a Bernoulli distribution [12]. The choice of Gelu is motivated by its full differentiability and continuity, facilitating effective backpropagation and optimization [13].

4.5.3. Dropout

Applied to the input features of Dense layers, dropout was incorporated in a gradual manner- the features of the first 2 layers were dropped by 50%, and those of the next 2 were dropped by 25%. This strategic implementation delays overfitting, enhancing the model's generalization.

4.5.4. Loss Function

The chosen loss function for the model is Mean Squared Error, measuring the square of the difference between expected and actual outputs. Ideal for regression problems, a lower MSE value indicates superior model performance.

4.5.5. Metric of Success

Mean Absolute Error (MAE) was selected as the success metric, representing the average absolute difference between predicted and actual values. Offering interpretability and robustness to outliers, MAE possesses a continuous derivative.

4.5.6. Optimizer

The chosen optimizer for the model is RMSProp, a variant similar to AdaGrad. RMSProp is designed to converge quickly by gradually adjusting the model’s learning rate, which is a scalar factor that indicates the speed of the gradient descent process. The learning rate refers to how quickly the optimizer changes the weights of a network in each step. AdaGrad calculates learning rates for each parameter based on squared gradients [14], but it suffers from a drawback known as rapid decay. This occurs as squared gradients accumulate, diminishing the learning rate over time and hindering further learning. RMSProp addresses this by employing a moving average of squared gradients, preventing the learning rate from diminishing too swiftly [15].

Require: Global learning rate η, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Initialize accumulation variables $\boldsymbol{r} = 0$
while Stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$.
Set $\boldsymbol{g} = \boldsymbol{0}$
for $i = 1$ to m **do**
Compute gradient: $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
end for
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho)\boldsymbol{g}^2$

Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\eta}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. % ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Figure 5. Algorithm for RMSProp optimizer. Learning rates of all parameters are set inversely proportional to an exponentially weighted moving average of squared partial derivatives over all training iterations. Taken from [17].

In Figure 5, *g* refers to the total gradient. RMSProp's approach involves computing the moving average of squared gradients over time, achieved through statistical calculations with subsets of the full dataset. Figure 5 illustrates the algorithm used. The moving average enables dynamic adjustment of learning rates for each parameter based on the gradient history, employing the principle of momentum to guide the model toward global minima.

By incorporating historically calculated gradients, RMSProp avoids getting stuck in local minima, facilitating the model's convergence to the global minimum. This concept draws inspiration from momentum in Physics, accelerating convergence. The Adam optimizer takes these principles further by integrating ideas from both RMSProp and momentum [16].

RMSProp's adaptive learning rate properties make it an excellent choice for regression tasks, aiding in the determination of optimal weights for parameters. As the learning rate is individually set for each parameter using the moving average, the optimizer ensures proper adjustments in case of excessively high or low gradients. This results in stable training and enhanced model optimization.

4.6. Model Training

Following the definition of parameters in section 4.5, the model was compiled, and the training loop was initiated.

4.6.1. Data Split

Prior to training, the dataset comprising 2,834 samples was divided into training, validation, and test sets. About 8% (234 samples) were reserved for the test set, while 20% (520 samples) constituted the validation set. The remaining data served as the training set. This separation is essential to ensure that the model, designed to learn patterns from the given data, is evaluated on unseen data during testing.

4.6.2. Learning Rate

The learning rate, a scalar determining the gradient descent speed, was specified as a parameter during model compilation. This initial learning rate guides the optimizer at the onset of training, dynamically adjusting throughout the process. RMSProp utilizes the moving average of squared gradients for this purpose. Setting an optimal initial learning rate is crucial, influencing the model's convergence speed. In this instance, the initial learning rate was configured at 0.01.

4.6.3. Number of Epochs

The number of epochs, initially set to 200, is considered adequate for the model to grasp data representations and establish a robust fit during training.

4.6.4. Batch Size

The model's batch size, set at 16, introduces a controlled level of noise to enhance generalization ability. A smaller batch size facilitates noise incorporation, contributing to improved model generalization.

4.7. Hyperparameter Tuning

Following the compilation with the previously defined parameters, adjustments to the hyperparameters were imperative for optimal performance.

4.7.1. Learning Rate

The initial training, utilizing the default learning rate, exhibited erratic performance metrics. To address this, the learning rate was systematically reduced to 0.001 and subsequently to 0.0001.

63

This adjustment proved pivotal in enabling proper model training. A larger learning rate risked overshooting optimal weight updates, causing random fluctuations in the loss function. The adoption of a smaller learning rate facilitated convergence towards the global optimum.

4.7.2. Model Size

In pursuit of overfitting during the model training phase, augmenting the model size emerged as a strategic approach. This involved elevating the number of layers or increasing the size of individual layers. Optimal model performance was achieved with a six-layer architecture: the first five layers each containing 384 units with GELU activation, and the final layer containing a single unit with no activation function to facilitate accurate predictions.

4.7.3. Number of Epochs

To induce overfitting, an alternative strategy involved prolonging the model training duration. The number of epochs was increased to 300 epochs, revealing distinct patterns in the model's performance.

4.7.4. Batch Size

Exploring the impact of batch size, adjustments were made to 32 and subsequently to 64. While a smaller batch size introduces beneficial noise for regularization, caution is warranted as it might contribute to overfitting by allowing the model to inadvertently learn the noise. Striking the right balance is essential for effective model training. The model performed best with the batch size as 64.

4.8. Results

The performance of the model is visualized through the loss function (MSE) and the performance metric (MAE) presented in Figures 6 and 7.

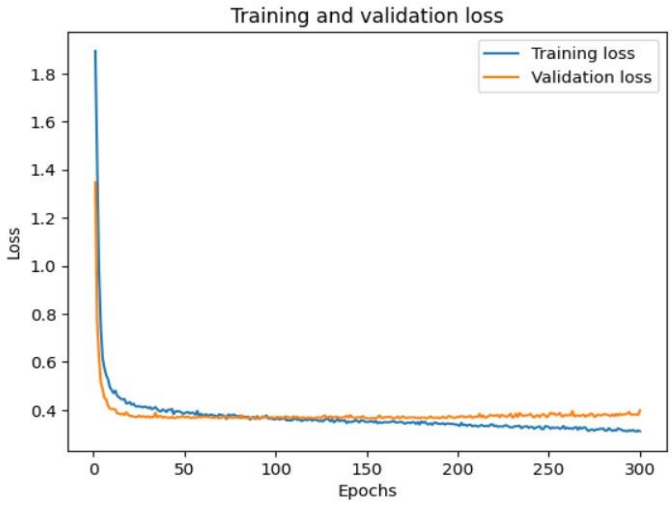

Figure 6. Training and Validation Loss

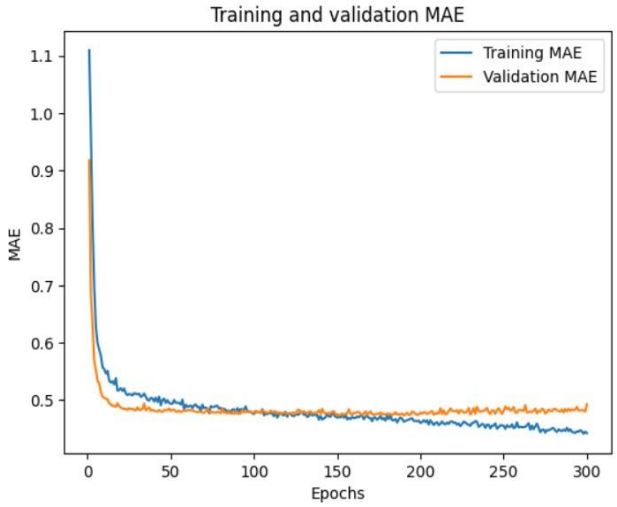

Figure 7. Training and Validation MAE

As depicted, the model attains commendable results, with both training and validation losses steadily decreasing over time, exhibiting signs of overfitting only after a sufficient number of epochs.

To better discern the validation MAE pattern, the initial high values for the first few epochs are omitted for clarity, as shown in Figure 8.

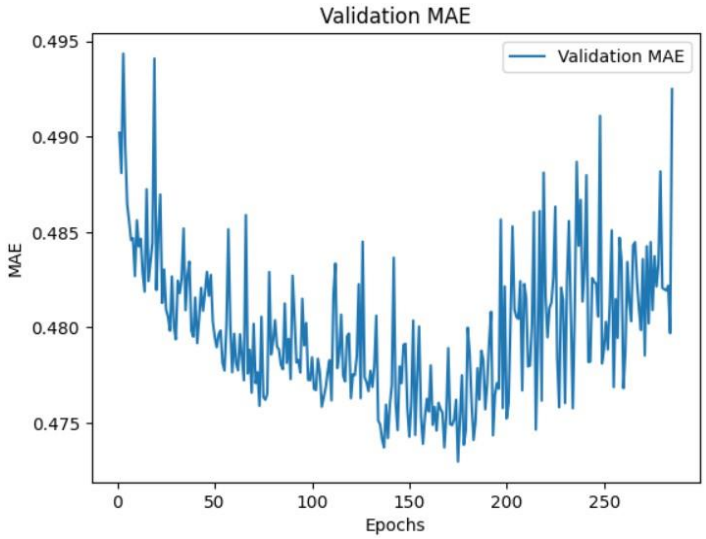

Figure 8. Validation MAE after truncating the first 15 epochs. Overfitting can be clearly seen.

The validation MAE reaches its lowest point at around 150 epochs, coinciding with the point where the training and validation losses intersect (Figure 7). Subsequently, while the training loss continues to decline, the validation loss starts to rise. This inflection point at 150 epochs signifies the attainment of a robust model fit.

At this juncture, the MAE for the model stands at approximately 0.472. This implies that the model's predictions deviate by 0.472 above or below the actual values. Considering the target column values range from -3.7 to 1.7, a mean absolute error of 0.472 is deemed acceptable.

4.8.1. Testing

Given the onset of overfitting around the mark of 150 epochs, a new training session was initiated, halted at 130 epochs, and the resulting model was employed to predict targets on the test data. The calculated mean average error for the predictions amounted to 0.53, slightly higher than the validation MAE. This behavior aligns with the typical performance degradation on unseen data, with the difference being inconspicuous.

The model exhibits a 10% relative error on the test data, implying an average deviation of 10% from the data range. While this level of error is deemed acceptable for this non-critical task, in practical applications, user awareness of potential errors is crucial, and user feedback can help gauge the impact on the overall experience.

5. CONCLUSION

This paper explores the development of a language model using the Transformer architecture and examines the techniques employed to prepare textual data for effective processing by a deep learning model. It emphasizes the Transformer architecture as an ideal choice for creating models that process textual data due to its effectiveness across various NLP tasks. Additionally, the paper highlights the importance of model design, particularly how the number and size of layers impact overall performance. Looking ahead, employing an ensemble of models presents a promising direction, potentially improving accuracy and overall outcomes.

In conclusion, this paper provides an overview of the current state of language technology, focusing on the Transformer model, while also considering future possibilities. Future work could explore using an ensemble of models and combining their predictions. This could improve the classification accuracy and robustness of the model, resulting in better generalization capability. Advancing the field will depend on exploring innovative model designs and fostering collaboration among different models.

REFERENCES

[1] Huang, Jui-Chan, Ko1, Kuo-Min, Shu, Ming-Hung & Bi-Min Hsu, (2020), "Application and comparison of several machine learning algorithms and their integration models in regression problems", *Neural Computing and Applications* 32, pp. 5461-5469.

[2] Krueger, David, Maharaj, Tegan, Kramár, János, Pezeshki, Mohammad, Ballas, Nicolas, Rosemary Ke, Nan, Goyal, Anirudh, Bengio, Yoshua, Courville, Aaron & Pal, Chris, (2016) "Zoneout: Regularizing rnns by randomly preserving hidden activations", *arXiv preprint arXiv:1606.01305*, doi:https://arxiv.org/abs/1606.01305

[3] Mikolov, Tomas, Chen, Kai, Corrado, Greg & Jeffrey Dean, (2013), "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, doi: https://arxiv.org/abs/1301.3781

[4] Jie, Liang, Jiahao, Chen, Xueqin, Zhang, Yue, Zhou & Jiajun, Lin, (2019), "One-hot encoding and convolutional neural network-based anomaly detection", *Journal of Tsinghua University (Science and Technology) 59*, no. 7, pp. 523-529.

[5] Wang, Shirui, Zhou, Wenan & Jiang, Chao, (2020), "A survey of word embeddings based on deep learning" *Computing 102*, pp. 717-740.

[6] Commonlit, May 3, 2021, "CommonLit Readability Prize", Kaggle.com, doi: https://www.kaggle.com/competitions/commonlitreadabilityprize/data

[7] TensorFlow (2015). Google. Available: https://www.tensorflow.org/

[8] Keras (2015). Francois Chollet. Available: https://keras.io/

[9] Chollet, Francois (2021) *Deep Learning with Python: Second Edition*. Manning Publications.

[10] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz & Polosukhin, Illia, (2017), "Attention is all you need.", *Advances in neural information processing systems 30*, pp. 6000 – 6010.

[11] Reimers, Nils & Gurevych, Iryna, (2019), "Sentence-bert: Sentence embeddings using Siamese Bert-Networks," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*. doi: https://arxiv.org/abs/1908.10084

[12] Hendrycks, Dan & Gimpel, Kevin, (2016), "Gaussian error linear units (gelus)." *arXiv preprint arXiv:1606.08415*. Doi: https://arxiv.org/abs/1606.08415

[13] Minhyeok Lee, (2023), "Mathematical analysis and performance evaluation of the gelu activation function in deep learning." *Journal of Mathematics*.

[14] Duchi, John, Hazan, Elad & Singer, Yoram, (2011), "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, doi: https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[15] Hinton, Geoffrey, Srivastava, Nitish & Swersky, Kevin. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 2012.

[16] Kingma, Diederik & Ba, Jimmy, (2014) "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*. Doi: https://arxiv.org/abs/1412.6980

[17] Goodfellow, Ian, Bengio, Yoshua & Courville, Aaron, (2017), *Deep learning*. Cambridge, MA: The MIT Press.