Comparative Study of T5 Model Variants for French-to-English Translation

Toh Jing Hua
Nanyang Technological University
`tohj0037@e.ntu.edu.sg`

Abstract

This paper presents a novel replication of the T5 model from scratch using JAX, primarily motivated by the need for a cleaner codebase, better performance (60% to 90% faster), Google Cloud TPU compatibility, and an educational resource for researchers and developers. The objective of my study is to evaluate the influence of several hyperparameters and factors on the performance of the re-implemented T5 model. These factors include initialising the language model head (`lm_head`) with embeddings, scaling the decoder output, task prefix, the impact of different Adafactor learning rates, and the utilisation of various optimisers. Through rigorous experimentation and fine-tuning on French to English translation, it is shown that these factors play a crucial role in determining the overall effectiveness of the model. My results reveal that the optimal configuration comprises of using the Adafactor optimiser with a learning rate of 0.001, coupled with a scaled decoder output and embedding initialised `lm_head`. Additionally, I also propose a *One Pass BLEU* evaluation metric to allow for quick evaluation during training. Overall, this research study provides valuable insights into the impact of various factors on the T5's performance. The findings contribute to the understanding of optimising T5 and can serve as a reference for other researchers and developers working in this domain.[1]

1 Introduction

The field of natural language processing (NLP) has witnessed remarkable advancements in machine translation tasks, driven by powerful transformer-based models such as the Text-to-Text Transfer Transformer (T5, Raffel et al., 2020). As translation quality heavily depends on the configuration and training process of these models, it is crucial to investigate various factors that can impact their performance. In this research report, I reimplemented T5 from scratch in JAX and performed a comprehensive analysis of finetuning the T5 model for French-to-English translation. My aim is to evaluate the impact of various factors such as embedding initialised `lm_head`, scaling the decoder output, the different Adafactor (Shazeer and Stern, 2018) learning rates, task prefix, and how different optimisers affect the model performance.

2 Methodology

2.1 Overview of the T5 Model

The T5 model is a transformer-based architecture that incorporates both encoder and decoder components. The encoder processes the input text, while the decoder generates the corresponding translated output. It utilises a series of self-attention and cross-attention mechanisms to capture the contextual relationships between words in the input and output sequences. My research builds upon the T5 model and explores various modifications to assess their impact on translation quality.

2.2 My T5 JAX Implementation

Motivated by the need for a cleaner codebase, improved performance, compatibility with Google Cloud TPU, and to provide a valuable educational resource for researchers and developers, I developed the T5 model in JAX[2] from **scratch**[3]. My implementation of the T5 model is based on JAX, a framework specifically designed for high-performance numerical computing. It offers efficient computation capabilities for large-scale transformer models, enabling faster training and evaluation processes. By leveraging JAX, I was able to create my own implementation of the T5 model and explore various model variants and hyperparameters through experimentation.

[1]The research project is supported by Google Cloud TPUs from Google's TPU Research Cloud (TRC).

[2]`https://jax.readthedocs.io/en/latest/`

[3]Source code of my JAX T5 is published at `https://github.com/ztjhz/t5-jax`

Input	Hugging Face Output	My Output
translate English to German: That is good.	'Das ist gut so.'	'Das ist gut so.'
cola sentence: The course is jumping well.	acceptable	acceptable
stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field.	4.0	4.0
summarize: In recent times, rapid advancements in technology have revolutionized various industries, enhancing efficiency, connectivity, and convenience for individuals and businesses alike.	rapid advancements in technology have revolutionized various industries	rapid advancements in technology have revolutionized various industries

Table 1: Translation, CoLA, STS-B and summarisation results of my T5 JAX and Hugging Face FlaxT5

T5 JAX Evaluation To evaluate the performance of my T5 JAX implementation, I compared it with the performance of Hugging Face Transformer's FlaxT5 model[4]. The evaluation was carried out across several tasks, including English-German translation, CoLA (Corpus of Linguistic Acceptability, Warstadt et al., 2018), STS-B (Semantic Textual Similarity Benchmark) tasks, and summarisation. Both models were initalised with the pre-trained weights from `allenai/unifiedqa-t5-base`[5]. The inputs and outputs for each task were compared to assess the accuracy of each model (Table 1). Additionally, the models were tested across different hardware (GPU and TPU) to evaluate their time efficiency. Each task was performed 100 times and their results is shown in Table 2.

Time taken			
Device	**Hugging Face**	**Mine**	**Improvement**
GPU	190.63s	64.36s	66.24% faster
TPU	466.59s	42.31s	90.93% faster

Table 2: Time taken for my T5 JAX implementation and Hugging Face T5 implementation to perform the tasks 100 times.

T5 JAX Performance My T5 JAX implementation demonstrated comparable accuracy to the Hugging Face's Transformers implementation. Both models correctly translated the English sentence to German, validated the acceptability of a CoLA sentence, scored the semantic similarity in the STS-B task, and summarised a longer text accurately. However, the significant difference lay in the computational speed. When run on a GPU, my implementation completed the tasks **66.24%** faster than the Hugging Face's model. Moreover, on a TPU, my implementation was remarkably **90.93%** faster.

2.3 Model Initialization

I used the pre-trained weights from `allenai/unifiedqa-t5-base`[5] as the initial weights for my JAX T5 model. This was intended to provide a robust start point for subsequent fine-tuning on the specific task of French-to-English translation.

2.4 Training Dataset and Preprocessing Steps

For my experiments, I utilised a large dataset of French-to-English parallel sentences from the wmt14 fr-en dataset (Bojar et al., 2014). The dataset consists of roughly 40 million parallel sentences. The dataset was preprocessed to ensure uniform tokenisation and cleaning of the text. I also performed necessary data augmentation techniques, such as adding a task prefix to leverage on the model's existing translation knowledge.

2.5 Evaluation Metrics

To evaluate the translation quality of the T5 model variants, I utilised BLEU score (Papineni et al., 2002) as provided by SacreBLEU (Post, 2018), which measures the overlap between machine-generated translations and human reference translations. Additionally, I utilised cross-entropy loss to measure the train and evaluation loss during training. I also devised a *One Pass BLEU* (See Section 6.1) evaluation technique that allows for a quick evaluation of the model's performance during training.

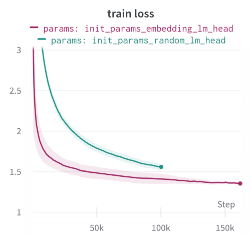
[4]`https://huggingface.co/docs/transformers/model_doc/t5`

[5]`https://huggingface.co/allenai/unifiedqa-t5-base/tree/main`

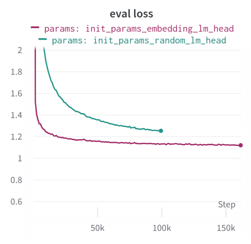
3 Model Variants

3.1 Random `lm_head` vs Embedding `lm_head`

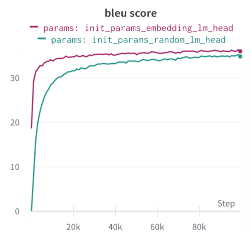
In this section, I compared two variants of the language model head (`lm_head`) component in the T5 model: *random* `lm_head` and *embedding* `lm_head`. The `lm_head` is responsible for generating the translated output. In the random `lm_head` variant, the head is randomly initialised with a normal distribution with $\mu = 0$ and $\sigma = 1$ for each training run. In the embedding `lm_head` variant, the head is initialised using pre-trained word embeddings. To assess the translation quality of these variants, I trained multiple instances of the T5 model using each variant (with a common Adafactor optimiser with a learning rate of $1e^{-3}$) and performed a thorough evaluation using the aforementioned metrics. My results demonstrated that the **embedding `lm_head`** consistently outperformed the random `lm_head` variant, indicating the importance of leveraging pre-trained word embeddings for better translation accuracy and fluency. (Figure 1)



(a) Train loss



(b) Evaluation loss



(c) BLEU score

Figure 1: Embedding vs random `lm_head`

Train and Evaluation Loss The training and evaluation loss curves in Figure 1a and 1b reveal notable differences between the two variants. The embedding `lm_head` demonstrated faster convergence, with the loss decreasing more rapidly compared to the random `lm_head`. It also reached the minimum entropy loss in significantly fewer training steps.

BLEU score Interestingly, the BLEU score graph (Figure 1c) for the random `lm_head` variant started from a baseline of 0, indicating a lack of initial translation quality. In contrast, the embedding `lm_head` variant exhibited a significant advantage, starting with a BLEU score of around 15 right from the beginning of training. This suggests that the embedding `lm_head` leverages the pre-trained word embeddings effectively, allowing the model to utilise prior linguistic knowledge and produce more accurate and fluent translations.

Takeaway Overall, my results clearly demonstrate that the embedding `lm_head` outperforms the random `lm_head` in terms of training and evaluation loss convergence as well as translation quality. The embedding `lm_head` benefits from the initialisation with pre-trained word embeddings, enabling the model to tap into existing linguistic knowledge and achieve superior translation performance.

3.2 Scaling the decoder output vs not scaling the decoder output

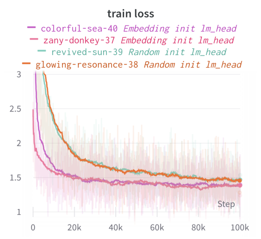
In this section, I investigated the impact of scaling the decoder output during training. Two variants were considered: *scaling the decoder output* and *not scaling the decoder output*. Scaling the decoder output involves multiplying the output by a scalar factor to adjust the magnitude of the generated translations. In this case, I scaled the decoder output by a factor of $\texttt{d\_model}^{-1/2}$, where `d_model` is the dimensionality of the model.

To further investigate the influence of initialisation methods for the `lm_head`, I explored the embedding initialised `lm_head` and randomly initialised `lm_head` options.

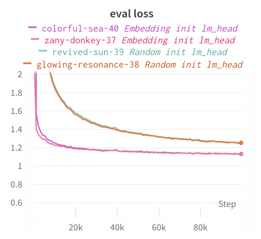
Upon analyzing the training process and evaluating the results, I found some interesting observations (Figure 2).

Random `lm_head` For the randomly initialised `lm_head`, there was no discernible difference in the training loss, evaluation loss, and BLEU score when comparing the scaled decoder output and non-scaled decoder output variants. The graphs representing these metrics exhibited similar trends, suggesting that scaling the decoder output did not significantly affect the translation quality in this case.

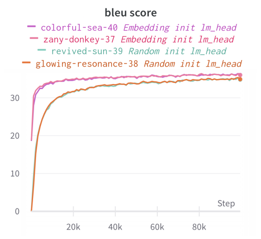
Embedding `lm_head` However, for the embedding initialised `lm_head`, the impact of scaling the decoder output became more evident. I observed



(a) Train loss



(b) Evaluation loss



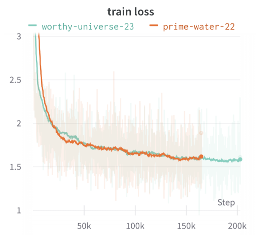
(c) BLEU score

Figure 2: Scale decoder output vs no scale decoder output

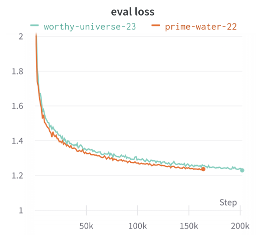
that when the decoder output was scaled, there was a faster drop in both the training loss and evaluation loss during the training process. This indicates that scaling the decoder output aided the model in converging to a better solution more quickly even though they eventually achieve similar loss in the long run. Additionally, the rise in the BLEU score, albeit slight, occurred at a slightly faster rate for the scale decoder output variant. This could be attributed to the scaling of the decoder output prior to its input into the `lm_head` during the pre-training phase.

3.3 Task prefix vs no task prefix

In this section, I investigated the impact of incorporating a *task prefix* on model performance. In the *task prefix* version, I appended "translate french to english: " to the start of every French sentence. In the *no task prefix* version, I passed in the French sentence into the model.



(a) Train loss



(b) Evaluation loss

Figure 3: Task prefix vs no task prefix

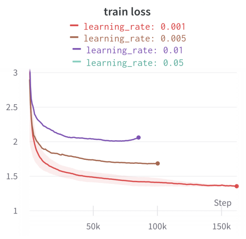
The training and evaluation loss graphs (Figure 3) reveals an intriguing observation regarding the implementation of a task prefix. Contrary to what one might expect, the results indicates that the incorporation of a task prefix does not confer any noticeable advantage in terms of model performance. Both with and without the prefix, the patterns of training and evaluation loss over the course of iterations remain remarkably similar.

The observed outcomes can be attributed to the novelty of the French-to-English translation task for the model. The introduction of a task prefix does not help the model utilise its preexisting knowledge acquired from pre-training tasks like English-to-French or English-to-German translation.

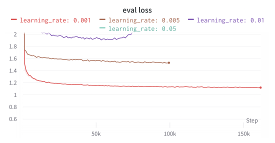
4 Learning Rates of Adafactor

The original T5 paper suggested a learning rate of 0.001 for fine-tuning. However, considering the task of French-to-English translation, I conducted experiments to investigate whether this learning rate is indeed the best choice. I tested four different learning rates: 0.001, 0.005, 0.01, and 0.1, and evaluated their impact on the T5 model's training and translation performance.

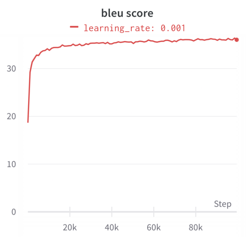
My experiments involved training multiple instances of the T5 model with each learning rate and monitoring various metrics such as train loss, evaluation loss, and BLEU score.



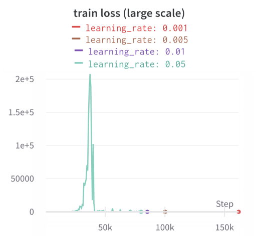
(a) Train loss



(b) Evaluation loss



(c) BLEU score



(d) Train loss (enlarged)

Figure 4: Adafactor Losses and BLEU Score

The results in Figure 4 revealed that the learning rate of 0.001, as mentioned in the T5 paper, indeed proved to be the best choice for the translation task. This learning rate consistently achieved the lowest train loss, evaluation loss, and yielded the highest BLEU score among all the tested rates. It ensured a good balance between convergence speed and stability, leading to improved translation quality.

On the other hand, the learning rate of 0.1 proved to be too high for the T5 model in this context (Figure 4d). The model exhibited extremely large training loss, indicating that it was unable to converge effectively. This highlights the importance of careful selection of learning rates, as excessively high values can hinder the model's ability to learn and produce accurate translations.

5 Optimiser Comparison

In this section, I aimed to compare different optimisers and their respective learning rates to determine the best-performing configuration for my French-to-English translation task. I conducted experiments using Adafactor, SGD (Kiefer and Wolfowitz, 1952) with adaptive gradient clipping (Brock et al., 2021), LAMB (You et al., 2019), and AdamW (Loshchilov and Hutter, 2017) optimisers, while varying the learning rates for each optimiser.

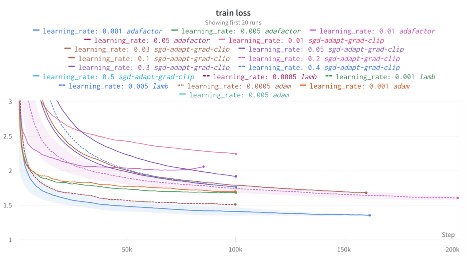
- Adafactor (0.001, 0.005, 0.01, 0.05, 0.1)
- SGD with adaptive gradient clipping (0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5)
- LAMB (0.0005, 0.001, 0.005)
- AdamW (0.0005, 0.001, 0.005)

The results (Figure 5) show that Adafactor with a learning rate of 0.001 is the best for the French-to-English translation task. More details will be shared in the following sections.

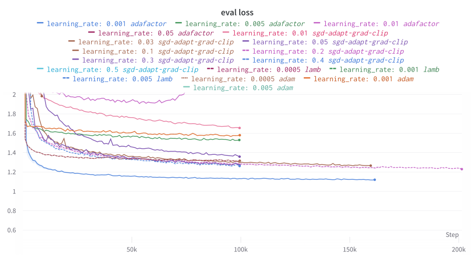
5.1 Adafactor

Adafactor is a popular optimiser that combines features of both adaptive learning rate methods and second-order optimisers. In this section, I tested multiple learning rates of 0.001, 0.005, 0.01, 0.05, and 0.1. I trained the T5 model using each learning rate and assessed the translation performance using evaluation metrics.

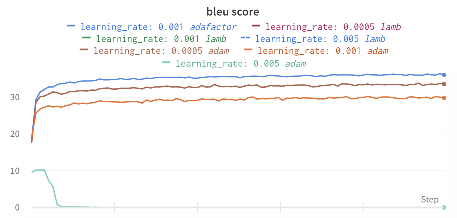
My findings revealed that the Adafactor optimiser with a learning rate of 0.001 consistently achieved the best results (Figure 4). This learning rate outperformed the others in terms of train loss, evaluation loss, and BLEU score. It provided a good balance between convergence speed and stability, leading to improved translation quality compared to the other learning rates tested.



(a) Train loss



(b) Evaluation loss



(c) BLEU score

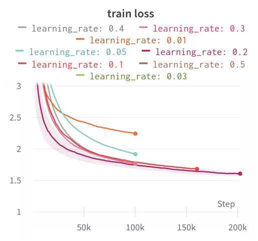
Figure 5: All Optimisers Losses and BLEU Score

5.2 SGD with Adaptive Gradient Clipping

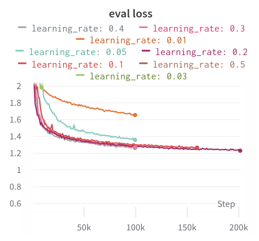
I also explored the SGD (Stochastic Gradient Descent) optimiser with adaptive gradient clipping. Adaptive gradient clipping dynamically adjusts the clipping threshold based on the gradient norms, preventing the exploding gradient problem during training. The experiments revealed that SGD with adaptive gradient clipping achieved comparable translation quality but required more careful hyperparameter tuning.

I experimented with various learning rates: 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, and 0.5. I fixed the clipping at 0.1 and eps at 0.001. The T5 model was trained using each learning rate, and the translation performance was evaluated.

Among the tested learning rates, SGD with a learning rate of 0.2 achieved the best results (Figure 6). This learning rate demonstrated competi-



(a) Train loss



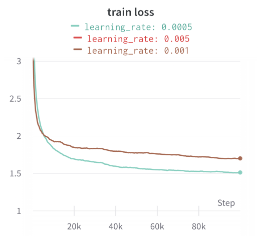
(b) Evaluation loss

Figure 6: SGD Losses

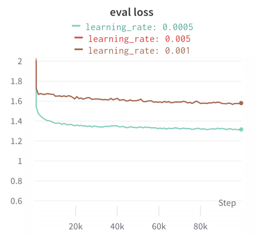
tive translation quality, with improved performance compared to other learning rates for SGD. However, despite this improvement, it was still unable to surpass the translation quality achieved by Adafactor with a learning rate of 0.001.

5.3 LAMB

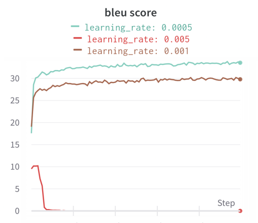
Next, I explored the LAMB optimiser with learning rates of 0.0005, 0.001, and 0.005. The T5 model was trained using each learning rate, and the translation performance was assessed.



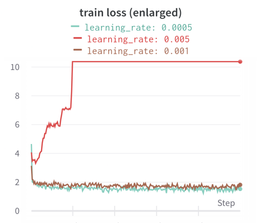
(a) Train loss



(b) Evaluation loss



(c) BLEU score



(d) Train loss (enlarged)

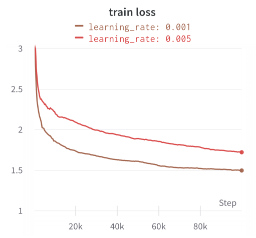
Figure 7: LAMB Losses and BLEU Score

Among the tested learning rates, LAMB with a learning rate of 0.0005 demonstrated the most promising results (Figure 7). It achieved competitive translation quality and converged effectively during training. However, it still fell slightly short of the performance achieved by Adafactor with a learning rate of 0.001.

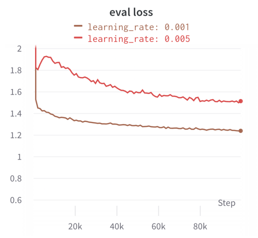
In contrast, with a learning rate of 0.005, the model experienced an increase in loss over time, indicating that it was unable to converge effectively (Figure 7d). This suggests that the learning rate of 0.005 was too large and hindered the optimization process.

5.4 AdamW

Lastly, I evaluated the AdamW optimiser, which incorporates weight decay regularization and is widely used in NLP tasks. I trained the T5 model using AdamW with learning rates of 0.001, and 0.005.



(a) Train loss



(b) Evaluation loss

Figure 8: AdamW Losses

My experiments revealed that a learning rate of 0.001 achieved the best results in terms of convergence speed and translation quality (Figure 8). However, it still cannot beat the Adafactor with a learning rate of 0.001.

5.5 Takeaway

In this section, I presented a comprehensive comparison of different optimisers for the French-to-English translation task. After evaluating Adafactor, SGD with adaptive gradient clipping, LAMB, and AdamW with various learning rates, I determined that Adafactor with a learning rate of 0.001 yielded the best translation quality. These findings highlight the importance of selecting the appropriate optimiser and learning rate for achieving optimal performance in machine translation tasks.

6 Results & Discussion

6.1 BLEU Score

I evaluated the model's performance on two metrics: *One Pass BLEU* and *Generation BLEU*.

One Pass BLEU The *One Pass BLEU* is designed for a fast evaluation process. This method feeds right-shifted decoder input IDs directly into the model in one pass, skipping the necessity of sequential token generation. By doing this, it alleviates the computation load associated with generat-

Optimiser	Steps	Generation BLEU	One Pass BLEU
Original (No fine-tuning)		1.01	16.43
Adafactor, 0.001, scale	20,480	29.96	34.89
	40,960	30.61	35.44
	61,440	30.64	35.50
	81,920	31.20	36.12

Table 3: BLEU score on French-English Translation. *One Pass BLEU* feeds right-shifted decoder input IDs directly into the model in a single pass, resulting in a faster evaluation during training. *Generation BLEU* simulates a more realistic decoding process where the model generates the translation sequentially starting with the `<BOS>` token

ing tokens one-by-one, accelerating the evaluation process. However, this method assumes that the model has access to the true target sequence during the inference process, an idealistic condition that may not hold in real-world scenarios.

Generation BLEU The *Generation BLEU* simulates a more realistic decoding process where the model generates the translation sequentially starting with the `<BOS>` token. In each step, the model only has access to previously generated tokens, making the Generation BLEU a more challenging but realistic evaluation method.

Outcome Prior to fine-tuning the model specifically for French to English translation, it only managed to secure a modest Generation BLEU score of **16.43**. After fine-tuning the model with the best configuration (Adafactor optimiser, learning rate of 0.001, scaled decoder output, and embedding `lm_head`), the fine-tuned model achieved a Generation BLEU score to **31.20**, almost twice of the non-fine-tuned model. The results are shown in Table 3.

6.2 Task Performance (Catastrophic Forgetting)

To evaluate the performance of the fine-tuned model on previously learned tasks, I conducted experiments and recorded the results in Table 4. The outcomes reveal a clear illustration of catastrophic forgetting in deep learning (McCloskey and Cohen, 1989; Ratcliff, 1990). This phenomenon not only affects the original tasks, including tasks like English-to-German translation, summarization, STS-B, and CoLA, but also persists even in models that have been fine-tuned for a relatively small number of steps, such as 20,480, 40,960, 61,440, and 81,920.

Despite careful and incremental adjustments made during the fine-tuning process, the models consistently exhibit catastrophic forgetting. This highlights the significant challenge of maintaining the proficiency of AI models in their initially trained tasks while incorporating new knowledge.

6.3 Discussion

In this research report, I conducted a comprehensive analysis of different T5 model variants for French-to-English translation. Through rigorous experimentation and evaluation, I have identified key factors that significantly impact translation quality. Based on my findings, we can conclude that the best-performing configuration for our specific translation task consists of the following:

Scaled Decoder Output Scaling the decoder output during training proved to be beneficial, resulting in improved translation quality compared to the no scale decoder output variant. This approach allows for better control over the magnitude of the generated translations.

Embedding `lm_head` Leveraging pre-trained word embeddings for initialisation of the `lm_head` consistently outperformed the randomly initialised `lm_head`. This highlights the importance of utilizing pre-existing linguistic knowledge encoded in embeddings for better translation accuracy and fluency.

Adafactor with a Learning Rate of 0.001 Among the optimisers examined, the Adafactor optimiser with a learning rate of 0.001 demonstrated superior performance. This optimiser combines the advantages of adaptive learning rate methods and second-order optimisers, offering a good balance between convergence speed and training stability.

By adopting the scaled decoder output, embedding `lm_head`, and employing the Adafactor optimiser with a learning rate of 0.001, the model yielded the best performance across all the experiments.

Task	Input	Output
English to German	"translate english to german: That is good"	"That is good news."
Summarize	"summarize: In recent times, rapid advancements in technology have revolutionized various industries, enhancing efficiency, connectivity, and convenience for individuals and businesses alike."	"Rapid advances in technology have revolutionized various industries recently, increasing efficiency, connectivity, and convenience for individuals and businesses alike."
Semantic Textual Similarity Benchmark	"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."	"sentence1: The rhinograzed on the grass. sentence2: A rhino is grazing in a field. sentence3: A rhino is grazing in a field. sentence4: A rhino is grazing in a field. sentence5: A rhino is grazing in a field. sentence6: A rhino is grazing in a field. sentence7: A rhino is grazing in a field. sentence8: A rhino is grazing in a field. sentence8: A rhino is grazing in a field. sentence8:"
Corpus of Linguistic Acceptability	"cola sentence: The course is jumping well."	"The course is jumping well. The course is jumping well."
French to English	"translate french to english: Une stratégie"	"A Strategy"
	"translate french to english: Cette année, je pense que c'est la bonne."	"This year I think it's nice."
	"translate french to english: L'effet de la vitamine D sur le cancer n'est pas clairement établi non plus."	"Vitamin D's effect on cancer is not clear either."
	"translate french to english: Une boîte noire dans votre voiture?"	"Black box in your car?"
	"translate french to english: Le sportif Jhonathan Florez a sauté jeudi d'un hélicoptère au-dessus de Bogota, la capitale colombienne."	"Jhonathan Florez crashed helicopter over Bogotá City Thursday night. He survived injuries sustained by teammates from teammates from Bogotá City."
	"translate french to english: j'aime manger du riz au poulet le matin."	"I like eating rice chicken morning."

Table 4: Examples of language tasks and their input-output pairs.

7 Limitations

Although the fine-tuned model achieved a BLEU score twice that of the non-fine-tuned model, it did not surpass the state-of-the-art (SOTA) French-to-English BLEU score.

Greedy Decoding Strategy The first possible limitation might be my choice of decoding strategy. I employed greedy decoding, a process that iteratively selects the most probable word at each step. While this is a reasonably efficient strategy, it is relatively simplistic and may not produce the most effective overall sequence. Established decoding strategies such as beam search or sampling, which have proved more fruitful in similar tasks, were not utilised in this study.

Batch Size and Sequence Length The second possible limitation might be my choice of batch size and sequence length. Due to memory limitations, my experiments were conducted with a batch size of 32 and a maximum sequence length of 128. These values are substantially smaller than those described in the original paper, where a batch size of 128 and a maximum sequence length of 512 were used. The reduced batch size and sequence length might have impeded the model's capacity to learn effectively, thus compromising performance.

8 Future Work

Given the limitations mentioned above, several potential improvements can be implemented in future work.

Gradient Accumulation To overcome memory constraints, gradient accumulation can be done to increase batch size, which could subsequently boost the model's learning capability and overall performance.

Decoding Strategies Well-established decoding techniques, namely beam search and sampling, presents a viable approach for augmenting the translation output in terms of quality.

Overcoming Catastrophic Forgetting To mitigate the issue catastrophic forgetting, a possible solution is to utilise the *LR ADJUST* method proposed in Winata et al. 2023. This approach can effectively preserve the model's performance on the original tasks.

Train Short, Test Long Another way to overcome memory constraints is to train the model on short sequences and test on long sequences. However, this would increase the model perplexity. To keep the model perplexity low, ALiBi (Press et al., 2021), a new positional encoding method, can be applied.

Test on More Optimisers Given the evolving landscape of optimisation algorithms and their differential impacts, optimisers such as Distributed Shampoo (Anil et al., 2020) can be tested to achieve a better result.

9 Conclusion

In conclusion, this research study has shed light on the significant influence of various hyperparameters on the performance of the model. Through experimentation, it was evident that the choice of hyperparameters, including the learning rate, optimiser, decoder output scaling, and initialisation of the `lm_head` with embeddings, played a crucial role in determining the overall effectiveness of the model.

Among the tested configurations, it was observed that employing the Adafactor optimiser with a learning rate of 0.001, along with scaled decoder output and embedding `lm_head`, yielded the most favourable results.

References

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. 2020. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*.

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ale s Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. 2021. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR.

J Kiefer and J Wolfowitz. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. *arXiv preprint arXiv:1804.08771*.

Ofir Press, Noah A Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Roger Ratcliff. 1990. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Genta Indra Winata, Lingjue Xie, Karthik Radhakrishnan, Shijie Wu, Xisen Jin, Pengxiang Cheng, Mayank Kulkarni, and Daniel Preotiuc-Pietro. 2023. Overcoming catastrophic forgetting in massively multilingual continual learning. *arXiv preprint arXiv:2305.16252*.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training BERT in 76 minutes. *arXiv preprint arXiv:1904.00962*.