Translation with Neural Networks

1. Introduction

Machine translation is a fundamental task in Natural Language Processing that enables communication across different languages, playing a critical role in areas such as global business, education, and digital content accessibility. Traditional translation systems often rely on hand-crafted rules and statistical methods, which struggle to adapt to the complexities of natural language. Recent advancements in deep learning have led to the development of powerful neural models that have significantly improved translation quality.

In this project, we focus on exploring, evaluating and fine-tuning existing neural network-based translation models to better understand their structure, behavior, and performance. Rather than making up new models from scratch, our work involves implementing and modifying pre-existing architectures such as Long Short-Term Memory networks, attention-based encoder-decoder models, and the Transformer model. These models are trained and tested on standard bilingual datasets, with measurable objectives including achieving benchmark translation quality, analyzing convergence during training, and comparing performance across model types.

The main goal of the project is to understand how different neural architectures handle the translation task and to assess how changes in model configurations, training parameters, and attention mechanisms impact performance. By doing so, we aim to draw insights into the strengths and limitations of each approach.

2. State-of-the-art

LSTM networks were introduced by Hochreiter and Schmidhuber to address the vanishing gradient problem in standard Recurrent Neural Networks. LSTMs use memory cells and gating mechanisms (input, output, and forget gates) to maintain long-range dependencies in sequential data. Their effectiveness has been demonstrated in various NLP tasks, including machine translation. However, a key limitation of LSTM models is that despite their ability to retain longer context compared to standard RNNs, they struggle with very long sequences.

To further address these challenges, Bahdanau et al. introduced the additive attention mechanism, allowing the decoder to dynamically focus on relevant parts of the input sequence at each decoding step. Similarly, Zhong et al. proposed the “Long Short-Term Attention” model, enhancing the ability to handle long or complex input sequences where key information may be dispersed throughout.

The Transformer architecture, introduced by Vaswani et al. in the seminal paper *"Attention Is All You Need"*, fundamentally changed the landscape of machine translation. Unlike RNN-based models, the Transformer relies entirely on self-attention mechanisms and forgoes recurrence altogether. This design allows for highly parallelized training, making Transformers significantly faster and more scalable than LSTMs. Despite its advantages, the

Transformer also has limitations. The model requires large amounts of data and computational resources for effective training.

3. Data Analysis

3.1 Transformer

For the Transformer model, we required a large and diverse dataset capable of supporting the extensive training needed by a model with millions of parameters. To meet this requirement, we used a high-volume English-Spanish parallel corpus containing approximately **13 million sentence pairs**. Although we did not utilize the full dataset due to computational constraints, a substantial portion was used for training and evaluation.

The dataset was relatively **balanced**, containing a mix of short and long sentences, which helped in training a model capable of handling varied input lengths. This diversity contributed positively to the model's generalization ability.
However, during the preprocessing phase, we identified a quality issue: **some sentence pairs contained empty English sentences paired with non-empty Spanish translations**. These misaligned entries were likely to introduce noise and hinder the training process. To resolve this, we implemented a simple **Python filtering function** to remove such examples.

3.2 LSTM

For the simple LSTM model, our initial approach involved using a **subset of 100,000 sentence pairs** from the original English-Spanish dataset. However, due to the LSTM's limited capacity to handle large vocabularies and long sequences, the results were suboptimal.

To address this, we experimented with **vocabulary size reduction**, aiming to simplify the input space. We attempted to filter the dataset to include only sentences composed of the **5,000 most frequent English words**. While this method significantly reduced the vocabulary complexity, it also **reduced the dataset size from 13 million to just 400 examples**, making it unsuitable for training.
As a solution, we sourced a different dataset: an **English-Italian parallel corpus** containing around **120,000 examples** and designed with a limited vocabulary of approximately 5,000 words. This dataset provided the right balance between size and complexity, making it more compatible with the LSTM architecture. The constrained vocabulary helped the model converge more easily and allowed us to obtain more interpretable results.

3.3 Seq2Seq + Attention

For the Sequence to Sequence with attention model, we chose to use a different **English-Spanish dataset** to diversify our experiments and observe how various data sources influence model performance. This dataset consisted of approximately **120,000 sentence pairs**, offering a moderate-sized corpus suitable for training an attention-based model without requiring excessive resources.

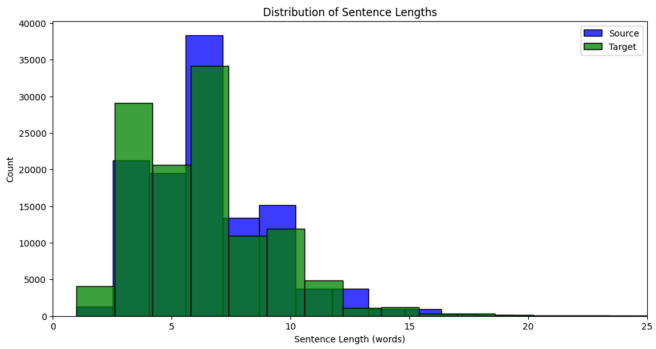

Figure 1:Sentence length distribution of the English-Spanish dataset

The plot show that most sentences in the dataset are short, mostly between 4 to 7 words. Very few sentences exceed 15 words, suggesting the dataset mainly contains simple, brief sentences. The most frequent words in both English and Spanish are common words like "I", "the", "you" and "que", "de", "el", indicating a natural language distribution but also highlighting a risk of the model overfitting to frequent function words.

4. Model and Optimization

4.1 Transformer

We focused on fine-tuning a pre-trained Transformer-based model to perform English-to-Spanish translation. We selected the T5-small model, a lightweight version of Google's Text-to-Text Transfer Transformer (T5), which is pre-trained on multiple translation tasks, including English-French and English-German. However, it was not optimized for English-Spanish translation, motivating our decision to fine-tune it on a targeted dataset.
The rationale for choosing T5-small included its balance between performance and computational cost, making it suitable for experimentation within limited hardware constraints, while still capturing the core capabilities of the full T5 architecture.

The T5 model is based on the Transformer encoder-decoder architecture from Vaswani et al. It has 6 encoder layers and 6 decoder layers. Each encoder layer includes multi-head self-attention, a feed-forward network, and layer normalization. Each decoder layer adds masked self-attention, cross-attention to the encoder output, and the same feed-forward and normalization components.
T5 uses shared embeddings and relative positional encodings, allowing efficient parameter reuse.

The model was trained using cross-entropy loss, comparing predicted tokens to the correct Spanish output at each step. This loss function encourages the model to prioritize correct sequences during decoding. However, in all cases we used BLEU to test the model's correctness.

These components work together to help the model learn complex language structures and meaning.

4.2 LSTM

The model follows a classic encoder-decoder framework.

The encoder is responsible for processing the input English sentence and encoding it into a context vector that summarizes its meaning. The components are as follows:

- Embedding Layer: Words are first passed through an embedding layer that maps each token to a dense vector of fixed size. This allows the model to learn semantic relationships between words.
- LSTM Layer: The embedded sequence is then passed through a single-layer LSTM. The encoder outputs the final hidden state and cell state, which are used to initialize the decoder.
- Sequence Reversal: The input sequence is reversed before being passed to the LSTM, a simple yet effective technique found to improve performance in early seq2seq models.

The decoder generates the translated Italian sentence, word by word, using the encoded context vector. It includes:
- Embedding Layer: Like the encoder, the decoder has its own embedding layer that learns representations for Italian vocabulary.
- LSTM Layer: The decoder LSTM is initialized with the encoder's final hidden and cell states. It processes one word at a time, either the previous predicted word or the correct word from the target sequence (teacher forcing).
- Linear Output Layer: A linear layer maps the decoder's LSTM output to a distribution over the Italian vocabulary. During inference, the most likely word is selected at each step using greedy decoding.

We use CrossEntropyLoss as the objective function, which is standard for multi-class classification tasks like language modeling.
The model is optimized using the AdamW optimizer for both the encoder and decoder. AdamW is a variant of the Adam optimizer that decouples weight decay from the gradient updates, leading to better generalization.

Both encoder and decoder optimizers are updated in each training iteration, and gradients are zeroed at the start of every batch.

4.3 Seq2Seq + Attention

The model is based on a Sequence-to-Sequence (Seq2Seq) architecture with Bahdanau Attention. The encoder uses Gated Recurrent Units (GRUs), selected over LSTMs for their balance between computational efficiency and the ability to model long-term dependencies. It begins with an embedding layer that converts input tokens (words) into dense vectors of size hidden_size. , followed by a GRU layer that generates hidden states. A dropout layer is applied after the embeddings to reduce overfitting. The encoder outputs all hidden states for use in attention, and the final hidden state initializes the decoder.

To address the bottleneck of traditional Seq2Seq models, Bahdanau's additive attention mechanism is implemented. At each decoding step, a score is computed between the decoder's current hidden state s_i and each encoder hidden state h_i, using the formula:

$e_{t,i} = v^T tanh(W_1 h_i + W_2 s_t)$, where W_1 and W_2 are learnable matrices that lineary projects h_i and s_i respectively into a new space and v is a learnable vector to reduce the result into a scalar. These scores are normalized using softmax to get the attention weights, used to tell the decoder where to focus at this step. Then, a context vector c_t is computed as the weighted sum of all encoder hidden states.

The decoder mirrors the encoder's structure, starting with an embedding layer, followed by the attention mechanism and a GRU layer that processes the concatenated context vector and embedded input. The output is passed through a linear layer projecting to the target vocabulary size. A dropout layer after embeddings further mitigates overfitting. Teacher Forcing is applied during training, feeding the true target token at each step to accelerate convergence.

5. Experiments

5.1 Transformer

Our initial approach involved fine-tuning the entire T5 model on a limited dataset of 5,000 English-Spanish sentence pairs. The rationale behind this choice was to perform a quick, low-cost test to assess the model's adaptability to the translation task. We set the learning rate to 5e-5 and trained the model for three epochs. However, the results were extremely poor. The model failed to learn meaningful alignments between English and Spanish, and output translations were essentially English.
Although these outcomes were not unexpected given the small data size, we increased the number of epochs to ten to evaluate whether extended exposure would yield any improvements. Unfortunately, the model still did not demonstrate any significant learning.

Encouraged by the potential of the model but aware of the limitations imposed by the small dataset, we expanded the training corpus to approximately 80,000 sentence pairs. Due to time constraints and the limitations of the Google Colab GPU environment, we only trained for 3 epochs. The model continued to perform poorly.

In an effort to reduce training time and potentially stabilize learning, we modified the model architecture by freezing the initial layers of both the encoder and the decoder. Our hypothesis was that early layers, which capture more general linguistic features, did not require fine-tuning and could be preserved. By freezing these layers, we hoped to reduce computational overhead and allow the model to focus on learning task-specific mappings in the later layers. We trained this variant for five epochs, which required approximately three hours. Unfortunately, the results remained unsatisfactory. The model still struggled to produce coherent translations, and performance did not meaningfully improve over previous attempts.

At this point, we returned to the full fine-tuning strategy, this time using a dataset of approximately 100,000 sentence pairs. We trained the model for five epochs, with training taking around five hours to complete. This experiment marked a slight turning point. Although the overall quality of translation was still poor, we observed that the model had begun to produce fragments of correct translations and occasionally translated individual words accurately. This suggested that learning was beginning to occur, albeit slowly and insufficiently.
These findings supported our intuition that the model was capable of learning the task but required either more training time or additional refinements to the setup to reach usable performance.

Building on the insights from previous trials, we reconsidered the role of the encoder. Since T5 had been pretrained on multiple languages, including English, we hypothesized that the encoder already had a robust understanding of English. As such, further fine-tuning of the encoder was likely redundant for our task, and possibly even detrimental due to overfitting risks.
In our final experiment, we froze the entire encoder and allowed training to focus exclusively on the decoder. Additionally, we adopted a more refined training schedule, incorporating a cosine learning rate scheduler with warm-up steps and weight decay to regularize training. The configuration included a learning rate of 5e-5 with cosine decay, a warm-up of 500 steps, a weight decay of 0.01, a batch size of 16 for both training and evaluation, a mixed precision training for speed and memory efficiency and five epochs of training with evaluation and checkpointing at each epoch
The model, at this stage, demonstrated some capacity to translate common or previously seen sentence patterns. However, for unseen sentences the translations were often incorrect or nonsensical. While this configuration yielded the best results among all our experiments, they were still underwhelming when compared to the capabilities of dedicated translation models.

Despite our efforts, the translation performance of the fine-tuned T5 model remained limited. These results suggest that general-purpose multilingual models like T5 require substantial amounts of data and training time to achieve strong performance on specific translation tasks. Moreover, without access to dedicated hardware or longer training schedules, performance is likely to plateau prematurely.

5.2 LSTM

All training runs shared a consistent setup: we used cross-entropy loss while ignoring padding tokens, and employed separate AdamW optimizers for the encoder and decoder. Training was conducted for ten epochs per experiment. The decoder used full teacher forcing, and loss was accumulated across the entire target sequence. Each batch's gradients were backpropagated and updated independently, with training and evaluation losses monitored periodically throughout.

We began by training the LSTM model on the same English-Spanish dataset previously used with the Transformer model, consisting of approximately 100,000 parallel sentence pairs. The training process proceeded without technical issues, but after several hours, it became apparent that the model was not learning effectively. The BLEU scores were very low, and a manual inspection of translations revealed that the outputs were largely incorrect, even for simple and frequently occurring phrases.
We attributed this poor performance primarily to the large vocabulary size of the dataset, which contained roughly 80,000 unique English tokens. Given the relatively small size and simplicity of the LSTM architecture, we believed that it was overwhelmed by the large search space and lacked the capacity to generalize meaningfully across such a broad lexical range. Motivated by this observation, we hypothesized that reducing the vocabulary size could significantly improve the model's performance. A smaller vocabulary would reduce the burden on the model's memory and increase the frequency of each token in the training data, potentially accelerating convergence and improving generalization for in-vocabulary tokens.

Our first attempt was to manually reduce the vocabulary of the original English-Spanish dataset by filtering out rare words or selecting only a subset of high-frequency examples. However, this proved challenging. The resulting dataset was too small and lacked sufficient diversity for meaningful training.
We then discovered a publicly available English-Italian dataset designed explicitly with vocabulary constraints. This dataset contained only 5,000 unique English words, providing a much more manageable lexicon for a low-capacity LSTM model. It consisted of high-quality sentence pairs where both sides were constructed using this restricted vocabulary.

We trained our LSTM model on this English-Italian dataset under the same conditions as before. The results were significantly improved. The model was able to produce nearly perfect translations on both training data and during ad hoc evaluation of new sentences composed from the same vocabulary set, and all that with only 10 minutes training. The training loss consistently decreased across epochs, indicating steady learning.
Given the nature of this dataset we determined that computing validation loss was unnecessary. The model was, by design, unable to generalize to unseen words, and thus performance evaluation was only meaningful within the context of the fixed 5,000-word vocabulary.

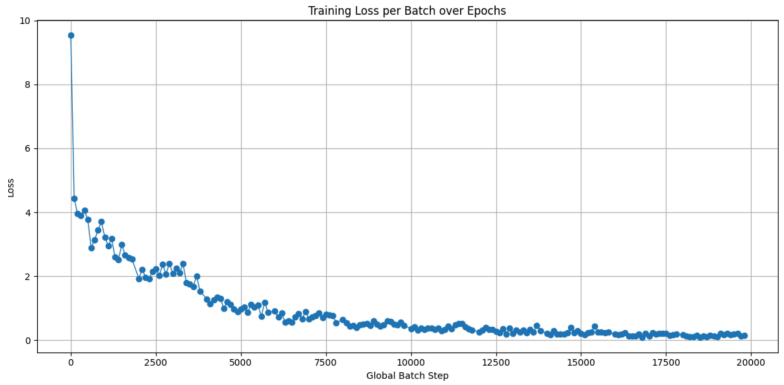

Figure 2: Training Loss per Batch over Epoch for LSTM model with small vocabulary dataset

These findings confirmed our hypothesis: a simple LSTM-based sequence-to-sequence model can achieve excellent performance, provided the input domain is sufficiently constrained. However, it also underscored the model's limitations: its utility is effectively restricted to closed-vocabulary scenarios. For real-world translation tasks requiring robust generalization across an open vocabulary, such an architecture would require additional components, such as attention or subword tokenization strategies.

5.3 Seq2Seq + Attention

We designed a set of experiments to evaluate the performance of the Seq2Seq model with Bahdanau Attention. As we couldn’t handle training the whole dataset, due to computational constraints, different dataset variations were tested:

Experiment 1: Training on the first 10,000 samples to see how the model acts.

Experiment 2: Filtering sentence pairs to a maximum of 7 tokens and trimming infrequent words from the vocabulary to reduce model size to reduce the total number of parameters and help manage computational requirements.
Experiment 3: Randomly sampling 60,000 sentence pairs to increase data diversity.
Experiment 4: Using pretrained English Word2Vec embeddings and training on a random 60,000 sample subset.

After preprocessing, the data was split into training and validation sets using an 90/10 split. Training used the Negative Log-Likelihood Loss (NLLLoss) and the Adam optimizer with a learning rate of 0.001. A batch size of 64 was used for efficiency. Dropout was applied into the embedding layers to mitigate overfitting by deactivating neurons with probability p=0.1 during training.
Model performance was evaluated using BLEU and METEOR scores on the validation set. BLEU assessed n-gram precision, while METEOR incorporated synonyms and word stems for a more semantic evaluation. Training and validation losses were monitored to detect overfitting. Table 1 summarizes the experiments, including the number of parameters, training time, epochs, and hidden size. Further experiments were done based on Experiment 2 by modifying hyperparameters:

Table 1: Experiments Summary

Experiment	Parameters	Hidden size	Epochs	Training time
Baseline (10k samples)	3.875.000	256	20	10 min
Filtered (max length 7, vocab trimmed)	4.297.131	256	20	1h
Random 60k Sample	12.628.083	256	30	7h
Random 60k Sample + Pretrained weights	15.019.923	300	10	5h

Table 2: Hyperparameter Tuning Based on Experiment 2

Changes	Motivation	Training Time
Incremented Dropout(0.1->0.3) and added weight decay(1e-4)	Experiment 2 had clear signs of overfitting	1h
Incremented hidden_size(256->512) and incremented epochs(20->30)	Reduce both training and validation loss	4.5h

Results:

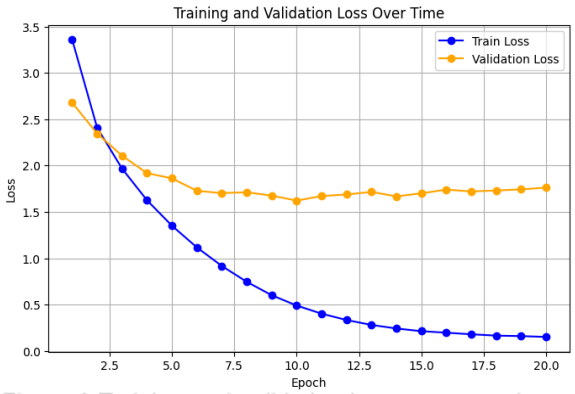

Figure 3:Training and validation loss over experiment 1

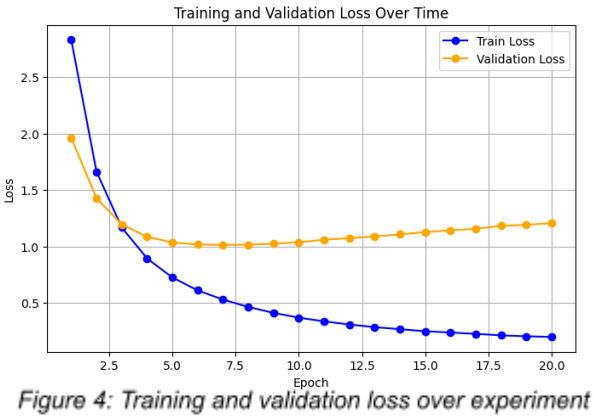

Figure 4: Training and validation loss over experiment 2 with weight decay and increased dropout

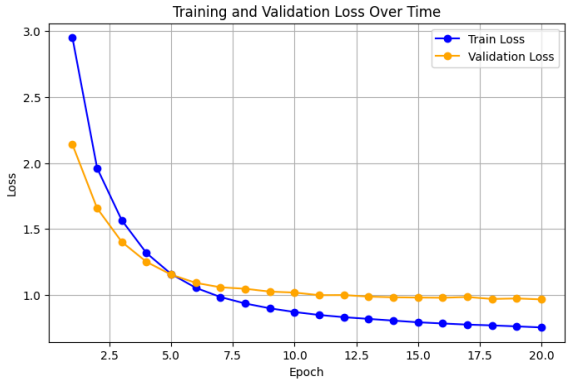

Figure 5: Training and validation loss over experiment 2 with weight decay and increased dropout

Figure 6: Training and validation loss over experiment 4

The validation loss curves in Experiment 1 suggest insufficient data for good generalization, while Experiment 2 indicates overfitting despite stable validation loss and decreasing training loss. To address this, regularization was enhanced by increasing dropout and adding weight decay, which reduced the gap between training and validation losses. Further improvements, including larger hidden size and more epochs, yielded only slight gains.

In Experiments 3 and 4, we also observed signs of overfitting, as the validation loss stabilizes while the training loss continues to decrease. Overfitting could potentially have been mitigated by applying additional regularization techniques, increasing the hidden size, and extending the number of training epochs. However, due to project time constraints and the high computational cost, we were unable to further explore these improvements.

Table 3: Evaluation Results:

Experiment	**Training Loss**	**Best Val Loss**	**BLEU**	**Meteor**
Baseline	0.1473	1.6217	0.1283	0.3762
Filtered	0.2091	1.0255	0.2401	0.5038
Filtered + Dropout 0.3 + Weight Decay	0.7551	0.9674	0.2965	0.5295
Filtered + Dropout 0.3 + Weight Decay + Hidden Size 512 + 30 Epochs	0.5808	0.9244	0.3002	0.5376
Random 60k Sample	0.1633	0.95	0.3612	0.5678
Random 60k Sample + Pretrained weights	0.2560	0.9821	0.3214	0.5509

The evaluation results show consistent improvement across experiments. The baseline model performed poorly, while dataset filtering dataset by sequence length and trimming the vocabulary improved performance. Introducing dropout and weight decay in this filtered setting further improved generalization, with a noticeable increase in evaluation metrics. Increasing hidden size and training epochs yielded further gains but at high computational cost. The best results were achieved with the random 60k sample, reaching a BLEU score of 0.36 and a METEOR score of 0.56, highlighting the importance of dataset size and training stability. Pretrained embeddings underperformed compared to the 60k sample, suggesting more training epochs are needed to realize their benefits.

6. Conclusions and Further work

While the final results from our Transformer experiments were underwhelming, we strongly believe we were on the right path. In later stages, the model began to show signs of learning, correctly translating some sentences. However, it is clear that more training time and a larger dataset were necessary to unlock its full potential.
Due to hardware limitations, we were constrained to shorter training runs on Google Colab. In the future, it would be valuable to access more powerful GPUs or to successfully set up training on our local GPU, something we attempted using Jupyter Notebook but encountered technical issues with. Resolving this would allow us to continue fine-tuning and fully assess the model's capabilities.

Our experiments with the LSTM-based sequence-to-sequence model demonstrated that even in the absence of an attention mechanism, it is still possible to achieve high-quality translations, provided that the problem space is sufficiently constrained. The strong performance observed on the limited-vocabulary English-Italian dataset suggests that the model possesses the foundational ability to learn translation mappings when the linguistic complexity is reduced.
This insight leads us to consider several potential directions for future work. First, we would like to further explore the limits of this simple architecture by training it on larger datasets for extended periods. While our initial attempts on a full-scale English-Spanish dataset yielded poor results, we suspect that with more extensive training time, a larger model size, and possibly improved regularization techniques, the LSTM could eventually learn to generalize across a broader vocabulary.

The experimental results on the Seq2Seq model with attention show a clear trend of improvement. The baseline achieved poor results, highlighting the need for a relatively large dataset and vocabulary to obtain decent performance. Applying filtering to the dataset significantly improved both validation loss and BLEU and METEOR scores. Further enhancements, including increasing dropout,adding weight decay, and incresing the hidden size with more training epochs, led to additional gains. The best BLEU score of 0.36 and a strong METEOR score of 0.56 were achieved with the filtered random 60k sample, demonstrating the importance of dataset size and careful regularization.

Pretrained word embeddings had decent results but did not outperform the random sample experiment in terms of score. This suggests that while pretrained embeddings enhance semantic similarity, they may require more training epochs to fully realize their benefits, as the experiment with pretrained embeddings was trained for only half the number of epochs.

Future work includes further hyperparameter tuning on the 60k dataset and the pretrained embeddings model, such as adjusting dropout, weight decay, and hidden size to optimize performance. For the pretrained embeddings model, extending the number of training epochs and experimenting with lower learning rates could also better leverage the embeddings potential. Additionally, introducing a learning rate scheduler and testing alternative optimization strategies for all strategies could further improve convergence and generalization.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.3215. https://arxiv.org/pdf/1409.3215

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. arXiv preprint arXiv:1706.03762. https://arxiv.org/pdf/1706.03762

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805. https://arxiv.org/pdf/1810.04805

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32. https://pytorch.org/docs/stable/index.html

Hugging Face. (n.d.). Transformers documentation. https://huggingface.co/docs

Hugging Face. (n.d.). Translation example notebook. https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/translation.ipynb

Řehůřek, R., & Sojka, P. (n.d.). Gensim: Word2Vec model documentation. https://radimrehurek.com/gensim/models/word2vec.html

StatQuest with Josh Starmer. (n.d.). StatQuest: Machine learning explained [YouTube channel]. YouTube. https://www.youtube.com/@statques

Donato Capitella. (n.d.). Deep learning tutorials [YouTube channel]. YouTube. https://www.youtube.com/@donatocapitella

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://www.bioinf.jku.at/publications/older/2604.pdf