

ISCTE
Iscte School of Technology and
Architecture (ISTA)



**Projeto final de Aprendizagem Automática
Avançada**

Elaborado por:

João Costa - 105530
Vicente Chã - 127688

Orientadores:

Professor Fernando Batista & Professor Sancho Oliveira

December 9, 2025

Contents

Contents	i
1 Introduction	1
2 Task 1	3
2.1 Task 1.1 – Comparative Report: LSTM-Based POS Tagger with Pre-trained Embeddings	3
2.1.1 Dataset and Preprocessing	3
2.1.2 Results: 300d GloVe LSTM	4
2.1.3 Results: 100d GloVe LSTM	4
2.1.4 Conclusion for Task 1.1	5
2.2 Task 1.2 — Transformer-Based Encoder Model (DistilBERT) . .	6
2.2.1 Model and Training	6
2.2.2 Results	6
2.3 Task 1.3 — LLM-Based POS Tagging	8
2.3.1 Procedure	8
2.3.2 Sentences	8
2.3.3 Results	8
2.3.4 Overall Conclusions	9
3 Task 2	13
3.1 Dataset Creation	13
3.2 Training a GPT Model	14
3.2.1 Training Performance	14
3.2.2 Qualitative Testing	14
3.3 Questions About the GPT Architecture	15
3.3.1 GELU Activation Approximation	15
3.3.2 Query Tensor Transformation	15
3.3.3 Attention Score Scaling	15
3.3.4 Layer Normalization	16
3.3.5 The MLP Block	16
3.3.6 Positional Embeddings	16
3.3.7 Top-k Filtering	17

4	Task 3	19
4.1	Model Architecture	19
4.1.1	Embedding Layer	19
4.1.2	LSTM Layer	19
4.1.3	Linear Layer (Decoder)	20
4.2	Training Methodology	20
4.2.1	Data Preparation	20
4.2.2	Loss Function and Optimization	20
4.2.3	Teacher Forcing	21
4.2.4	Training Results	21
4.3	Evaluation	21
4.3.1	Error Analysis	22
5	Conclusions	23
5.1	Technologies Used	23
5.2	Conclusion	23

Chapter

1

Introduction

This report documents the implementation and analysis of deep learning architectures for sequence processing tasks. The project explores the transition from recurrent neural networks to attention-based models across three distinct tasks:

- **Task 1: POS Tagging.** We benchmark Long Short-Term Memory (LSTM) networks against pre-trained Transformer encoders on the Universal Dependencies English Web Treebank dataset.
- **Task 2: GPT Adder.** We train a decoder-only Transformer (minGPT) to perform arithmetic addition, including a theoretical analysis of the architecture's components.
- **Task 3: RNN Adder.** We implement a custom Recurrent Neural Network to solve the same arithmetic task.

Chapter

2

Task 1

2.1 Task 1.1 – Comparative Report: LSTM-Based POS Tagger with Pre-trained Embeddings

In this task, a sequence model LSTM was trained for Part-of-Speech (POS) tagging using the provided training set. The model incorporated a pre-trained embeddings GloVe layer during development. A classifier was employed to generate one POS tag per token.

Model performance was evaluated on the test set, and both overall accuracy and per-class precision, recall, and F1-score were reported.

2.1.1 Dataset and Preprocessing

The experiments used the UD English EWT dataset, containing:

- 12,544 training sentences
- 2,001 development sentences
- 2,077 test sentences
- Vocabulary size: 20,203
- POS tagset size: 19

Three BiLSTM models were trained using frozen GloVe embeddings with dimensions 100d, 200d, and 300d to study the effect of embedding dimensionality on training time and accuracy. Training was performed for 10 epochs using cross-entropy loss and the Adam optimizer on GPU hardware. The model produces one POS tag per token.

2.1.2 Results: 300d GloVe LSTM

- Parameters: 6,506,103
- Training time: 26.30 s
- Test accuracy: 92.94%

The loss curve in Fig. 2.1 shows a rapid drop in training loss during the first two epochs, while the validation loss stabilises around epoch 3. After this point, training loss continues to decrease monotonically, but the validation loss slowly increases, indicating mild overfitting. This pattern is consistent with the model's relatively large parameter count, which allows it to fit the training data easily but provides limited generalisation benefits over smaller variants.

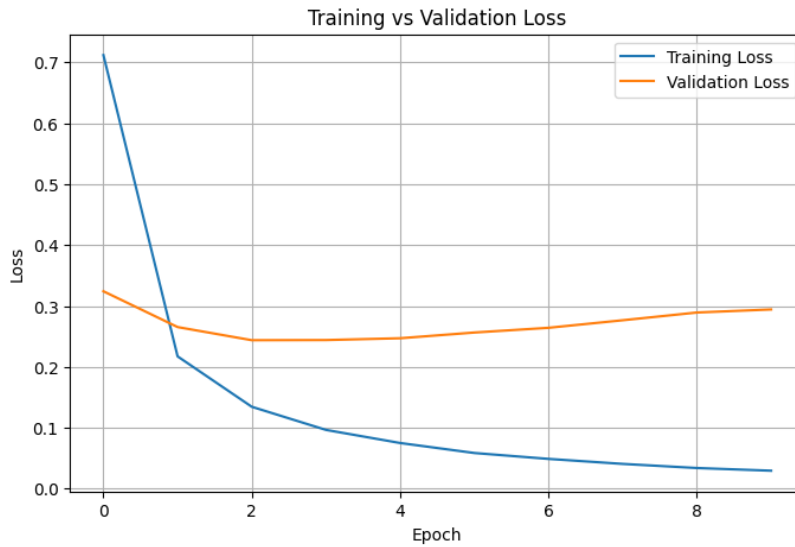


Figure 2.1: Training and Validation Loss over 10 Epochs for the 300d LSTM. The widening gap between curves from epoch 3 onward indicates overfitting.

Tag-level performance shows very strong results on high-frequency tags such as ADP, AUX, DET, PART, PRON, and PUNCT. Lower performance occurred for PROP (F1 = 0.78), NUM (0.83), SYM (0.85), and especially X (F1 = 0.04), where the extremely small support severely limits learnability.

2.1.3 Results: 100d GloVe LSTM

- Parameters: 4,383,403

- Training time: 21.54 s
- Test accuracy: 92.92%

The training curve in Fig. 2.2 shows the most stable behaviour among the three LSTM models. Validation loss decreases for the first three epochs and then plateaus with only minor fluctuations, indicating significantly less over-fitting than in the 300d model. Despite having fewer parameters, the 100d model attains the highest test accuracy (92.99% in the aggregate comparison table), demonstrating that increasing embedding size does not translate into better performance for this POS task.

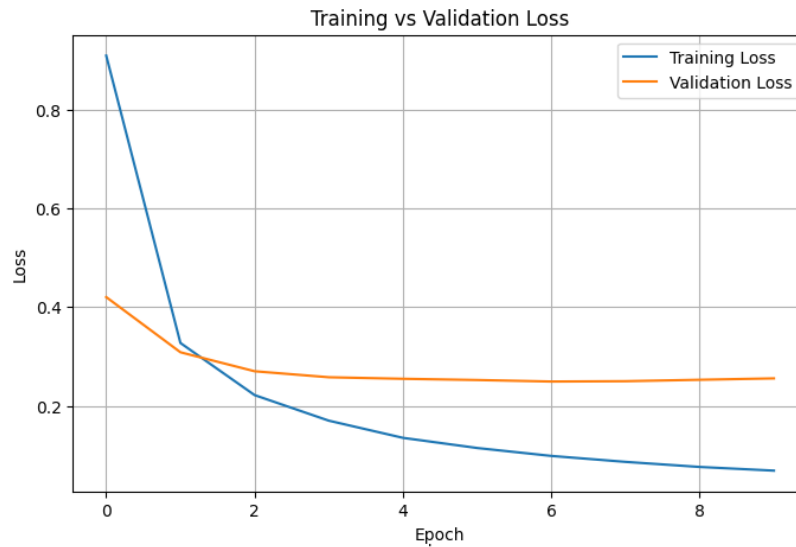


Figure 2.2: Training and Validation Loss over 10 Epochs for the 100d LSTM. The relatively small gap between curves indicates good generalisation.

2.1.4 Conclusion for Task 1.1

All three LSTM architectures achieved approximately 93% accuracy. As expected, the 100d model offered the best balance between performance and efficiency, training fastest while achieving slightly higher accuracy than the larger variants. Increasing embedding dimensionality increased training time and model size but did not improve generalisation. Across all models, persistent difficulties were observed for PROP, SYM, and X, mainly due to low support and semantic ambiguity.

Model	Params	Acc.	Train Time (s)	Macro Prec.	Macro Rec.	Macro F1
LSTM 100d	2.26M	92.99%	16.34	0.90	0.88	0.89
LSTM 200d	4.38M	92.92%	21.54	0.89	0.86	0.87
LSTM 300d	6.51M	92.94%	26.30	0.89	0.86	0.88

Table 2.1: Comparison of LSTM models trained with frozen GloVe embeddings of different dimensionalities.

2.2 Task 1.2 — Transformer-Based Encoder Model (DistilBERT)

2.2.1 Model and Training

A DistilBERT encoder was fine-tuned for token classification. The alignment of subwords to tokens assigned labels only to the first subword of each token.

- Base model: DistilBERT
- Number of parameters: ~65M
- Early stopping triggered at epoch 3
- Training time: 529.9 s

The training curve exhibits very rapid convergence: the validation loss decreases substantially as early as the first epoch, with only a marginal improvement observed in the second. By the third epoch, the validation loss begins to increase slightly, while the training loss continues to decrease. This pattern indicates the onset of overfitting.

Consequently, early stopping was triggered, interrupting the training process after three epochs, which represents the optimal stopping point. This behavior confirms that pre-trained transformer models adapt quickly to the task with only a few iterations, thereby avoiding unnecessary training.

2.2.2 Results

DistilBERT showed excellent performance in AUX, DET, PART, PRON, VERB, and PUNCT. The challenging tags included SYM and X (F1 = 0.44), and some confusion occurred between NOUN and PROP.

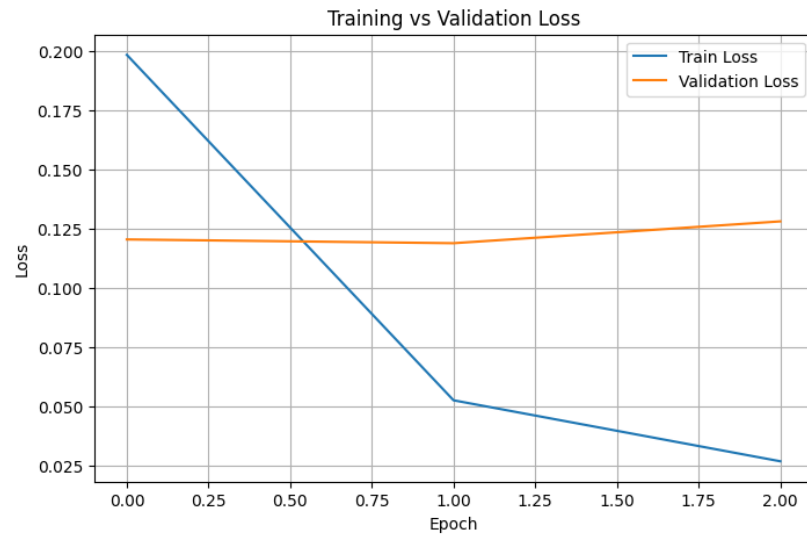


Figure 2.3: Training and Validation Loss over 3 Epochs

POS Tag	Precision	Recall	F1-score	Support
ADJ	0.92	0.96	0.94	1788
ADP	0.98	0.97	0.98	2025
ADV	0.96	0.94	0.95	1191
AUX	1.00	0.99	1.00	1543
CCONJ	1.00	1.00	1.00	736
DET	0.99	0.99	0.99	1897
INTJ	0.89	0.89	0.89	121
NOUN	0.96	0.93	0.94	4123
NUM	0.96	0.98	0.97	542
PART	1.00	0.99	0.99	649
PRON	0.99	0.99	0.99	2165
PROPN	0.89	0.92	0.91	2075
PUNCT	0.99	1.00	0.99	3096
SCONJ	0.94	0.96	0.95	384
SYM	0.85	0.83	0.84	113
VERB	0.98	0.98	0.98	2606
X	0.46	0.43	0.44	42
–	0.96	0.99	0.97	354
Accuracy		0.9667		25450
Macro Avg	0.93	0.93	0.93	25450
Weighted Avg	0.97	0.97	0.97	25450

Table 2.2: Performance of the DistilBERT transformer model on the test set. Training time: 529.89 seconds.

2.3 Task 1.3 — LLM-Based POS Tagging

2.3.1 Procedure

An LLM (ChatGPT) was queried using a structured prompt enforcing:

- Clear explanation of the POS tagging task
- Restriction to the allowed POS tagset
- One tag per token, in order, without additional text

Ten examples sentences from the test set were evaluated.

2.3.2 Sentences

The sentences provided by the LLM are described below.

- 10 test sentences
- 107 Words
- POS tagset size: 16

2.3.3 Results

LSTM baseline on the examples: 92.52% accuracy, macro F1 = 0.88

DistilBERT on the examples: 96.26% accuracy, macro F1 = 0.92

Common confusions were observed across all models. The most frequent involved distinctions between NOUN and PROPN, as well as between DET and ADJ. The models also struggled with NOUN/AUX/VERB ambiguities, particularly for tokens such as meeting or being, which can assume multiple syntactic roles depending on context. Additionally, performance on rare tags, such as SYM and X, was consistently weaker due to their low representation in the training data.

Transformers significantly reduced these errors, particularly for long or complex syntactic sentences. LSTMs struggled more with out-of-vocabulary items and lacked deep contextual information. LLMs performed well, but sometimes produced inconsistent outputs depending on prompt phrasing.

Tag	LSTM (Acc = 92.52%)			DistilBERT (Acc = 96.26%)			Support
	Prec.	Rec.	F1	Prec.	Rec.	F1	
ADJ	0.86	1.00	0.92	0.86	1.00	0.92	6
ADP	0.88	1.00	0.93	1.00	1.00	1.00	7
ADV	1.00	1.00	1.00	1.00	1.00	1.00	3
AUX	1.00	1.00	1.00	1.00	1.00	1.00	4
CCONJ	1.00	0.50	0.67	1.00	1.00	1.00	2
DET	1.00	0.93	0.97	1.00	0.93	0.97	15
INTJ	1.00	1.00	1.00	1.00	1.00	1.00	1
NOUN	0.79	0.95	0.86	0.87	1.00	0.93	20
NUM	1.00	1.00	1.00	1.00	1.00	1.00	2
PART	1.00	1.00	1.00	1.00	1.00	1.00	2
PRON	1.00	1.00	1.00	1.00	1.00	1.00	4
PROPN	0.86	0.75	0.80	1.00	0.75	0.86	8
PUNCT	1.00	1.00	1.00	1.00	1.00	1.00	13
SCONJ	1.00	1.00	1.00	1.00	1.00	1.00	2
SYM	0.00	0.00	0.00	0.00	0.00	0.00	1
VERB	1.00	0.88	0.94	1.00	1.00	1.00	17
Macro Avg	0.90	0.88	0.88	0.92	0.92	0.92	107
Weighted Avg	0.92	0.93	0.92	0.96	0.96	0.96	107

Table 2.3: Comparison of LSTM vs. DistilBERT on the 10-sentence evaluation subset. DistilBERT improves performance across most categories, especially NOUN, PROPN, and VERB.

Model	Accuracy	Training Time	Notes
LSTM 300d	92.94%	26.3 s	Slowest, no accuracy benefit
DistilBERT	96.67%	530 s	Best model (+3.7% over LSTM)

Table 2.4: Performance and efficiency comparison across models. [†]Evaluated on a small subset only.

2.3.4 Overall Conclusions

The DistilBERT model provided the best POS tagging performance, outperforming LSTM models by approximately 3–4 percentage points in accuracy. However, LSTMs remain competitive considering their far lower computational cost. LLMs demonstrate strong zero-shot performance but lack the reliability of fully fine-tuned models. Tag-level analysis shows that rare tags (SYM, X) and the NOUN/PROPN boundary remain challenging across architectures.

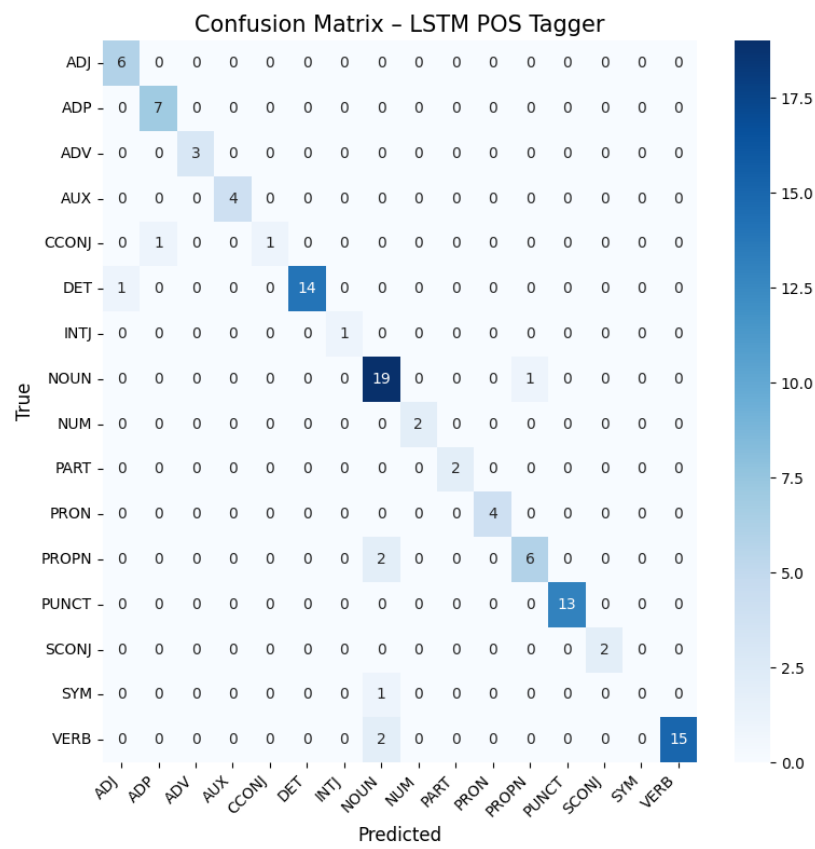


Figure 2.4: Training and Validation Loss over 3 Epochs early-stopping

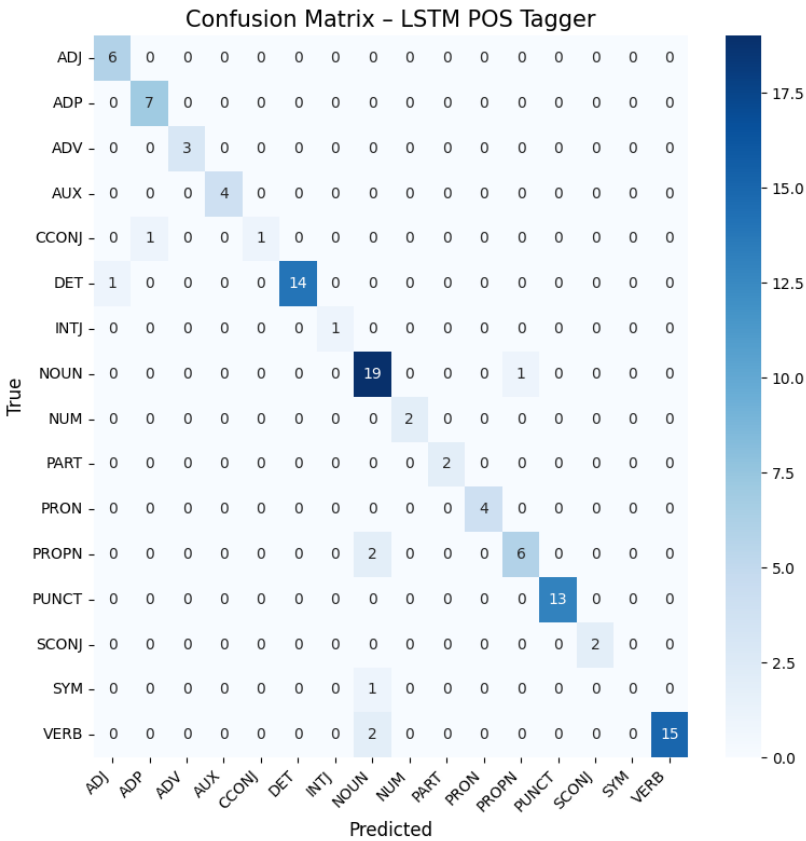


Figure 2.5: Training and Validation Loss over 3 Epochs early-stopping

Chapter

3

Task 2

In this section of the project, the objective was to train a simpler version of a Transformer neural network architecture, the minGPT, to perform simple addition problems involving two integers of up to two digits.

3.1 Dataset Creation

The first step involved generating and preprocessing a dataset of arithmetic expressions. As per the requirements, we generated expressions involving two operands, where each operand is a non-negative integer with a maximum of two digits.

The dataset consists of character strings representing the equation, such as '20+30=50.'. To standardize the input for the neural network, the following preprocessing steps were implemented:

- **Vocabulary Mapping:** We defined a character-to-index mapping including digits 0–9 and special tokens: '+' (10), '=' (11), '.' (12, end token), and '#' (13, padding token).
- **End Token:** A token '.' was appended to the end of every expression to signal the termination of generation to the transformer.
- **Padding:** Since addition expressions vary in length, a padding token '#' was used to right-pad all sequences to a fixed length based on the longest possible expression supported by the logic.

To facilitate autoregressive learning, we constructed the training pairs from the tokenized data. The inputs consisted of the full tokenized equations, while

the targets were also the full tokenized equations except the very first token to train the model on next-token prediction.

3.2 Training a GPT Model

We trained a decoder-only GPT model on the generated dataset. The model configuration included 3 layers, 3 attention heads, and an embedding dimension of 48.

3.2.1 Training Performance

The model was trained for 5,000 iterations using the AdamW optimizer. Throughout the optimization process, the loss demonstrated a consistent decline while accuracy steadily improved.

Starting with a loss of approximately 1.54 at iteration 100, the model improved significantly by iteration 2,000, where the loss dropped to 1.02. At this stage, the model attained a training accuracy of 61.10% and a validation accuracy of 60.10%. By the final iteration, the model converged to a loss of approximately 0.85, achieving a final training accuracy of 98.37% and a validation accuracy of 98.30%. The minimal divergence between training and validation metrics suggests robust generalization.

3.2.2 Qualitative Testing

After training, we evaluated the model qualitatively by prompting it with incomplete addition expressions. Since these kinds of models are stateless, the model sometimes predicted the correct sums and terminated generation with the appropriate tokens, but other times failed to generate the correct tokens. Examples of the model output variations (all of these were generated by the same model):

Iter 1:	Prompt: $1+1=$	Model output: $1+1=2.##$
	Prompt: $12+34=$	Model output: $12+34=46.#$
	Prompt: $5+6=$	Model output: $5+6=10.#$
	Prompt: $99+1=$	Model output: $99+1=100.$
	Prompt: $8+9=$	Model output: $8+9=17.#$
Iter 2:	Prompt: $1+1=$	Model output: $1+1=3.##$
	Prompt: $12+34=$	Model output: $12+34=45.#$
	Prompt: $5+6=$	Model output: $5+6=9.##$
	Prompt: $99+1=$	Model output: $99+1=100.$

	Prompt: 8+9=	Model output: 8+9=179.
Iter 3:	Prompt: 1+1=	Model output: 1+1=2.##
	Prompt: 12+34=	Model output: 12+34=46.#
	Prompt: 5+6=	Model output: 5+6=12.#
	Prompt: 99+1=	Model output: 99+1=900.
	Prompt: 8+9=	Model output: 8+9=159.

These results confirm that the character-level GPT successfully learned the rules of addition and the syntax of the provided dataset but can still make unpredictable mistakes.

3.3 Questions About the GPT Architecture

The following section addresses specific questions regarding the minGPT implementation used in this project.

3.3.1 GELU Activation Approximation

The NewGELU class computes an approximation of the Gaussian Error Linear Unit (GELU) rather than the exact definition. This is done to save computation time and resources because the exact GELU requires computing the cumulative distribution function of a Gaussian using the error function, which is computationally expensive. In contrast, the approximation uses the tanh function, which only requires basic mathematical operations.

3.3.2 Query Tensor Transformation

The line `q = q.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)` transforms the query tensor from a single large embedding vector into smaller separate sections for each attention head. This allows the model to process the input in parallel by grouping the data into segments by head rather than by word.

3.3.3 Attention Score Scaling

- A:** The attention score division acts as a normalization factor for the Q and K dot product, if their dimension is large the dot product can grow very large.
- B:** If the scaling were to be removed the dot products could become extremely large which causes several problems.

- C:** This line applies a mask to the attention scores so that the model can't look into the future unseen tokens and know their values.
- D:** If the mask were omitted, the model would be able to see into the future and the model would learn from the attention mechanism since it would already know the next tokens in line and would only learn to use the tokens ahead of the current one and would fail on the testing phase since it was trained with unobtainable data.

3.3.4 Layer Normalization

- A:** No, the original implementation uses Post normalization and the `minGPT` uses Pre normalization.
- B:** The key difference is where the layer of normalization is placed relative to the residual connection, the attention layer and MLP layer. On the Post normalization the normalization is applied after the residual addition $x = \text{Norm}(x + \text{SubLayer}(x))$, and in the Pre normalization the normalization is applied before the sublayer input, inside the residual branch $x = x + \text{SubLayer}(\text{Norm}(x))$. Post norm can cause vanishing gradient or exploding gradients in deep networks and might need tweaks on the learning rate, the Pre norm improves stability since there is a clear path for the gradient to follow, this improves the deep networks problem.

3.3.5 The MLP Block

The two-layer MLP serves as the processing part of the transformer for each token, whereas the attention layer handles communication between tokens. The dimensionality is expanded to $4 \times n_{embd}$ to project the data into a larger space. This expansion allows the transformer to learn more complex patterns and relationships before the data is compressed back down for the next block.

3.3.6 Positional Embeddings

Positional embeddings are added element wise to the token embeddings at the input stage. They are necessary because the self attention mechanism needs the order information, it does not know the order of tokens. These embeddings inject sequence information, allowing the model to understand word order and understand the meanings between different sets of identical tokens.

3.3.7 Top-k Filtering

This line isolates the top k most likely tokens to be correct by finding the k highest score in the current distribution of tokens. Then, it deletes all other tokens by setting their score to negative infinity, so that when softmax is applied they won't be chosen.

Chapter

4

Task 3

The objective of this task is to develop a Recurrent Neural Network (RNN) capable of performing simple addition problems.

4.1 Model Architecture

We implemented a custom RNN module, which processes input sequences through three primary stages: an embedding layer, an LSTM layer, and a linear output layer.

4.1.1 Embedding Layer

The input to the network consists of batches of integer coded character sequences, taken from the same dataset that was used in Chapter 3. The vocabulary is all digits (0-9), plus the symbols '+', '=', '.', and the padding token '#'.

- **Input:** Batch of shape $[N, L]$, where N is the batch size (64) and L is the sequence length.
- **Layer:** `nn.Embedding(vocab_size, hidden_size)`
- **Output:** Batch of shape $[N, L, H]$, where H is the hidden size (128).

4.1.2 LSTM Layer

The core of the sequence processing is a Long Short-Term Memory (LSTM) layer. This layer maintains a hidden state that evolves as it processes the se-

quence step-by-step, allowing the model to capture dependencies between the operands and the resulting sum.

- **Layer:** `nn.LSTM(hidden_size, hidden_size, batch_first=True)`
- **Function:** It processes the embedded vectors and outputs a sequence of hidden states, one for each time step.
- **Output:** Batch of shape $[N, L, H]$.

4.1.3 Linear Layer (Decoder)

The final stage is a fully connected linear layer that maps the LSTM hidden states at each time step back to the vocabulary space, producing logits (un-normalized prediction scores) for the next character.

- **Layer:** `nn.Linear(hidden_size, vocab_size)`
- **Output:** Batch of shape $[N, L, V]$, where V is the vocabulary size.

4.2 Training Methodology

4.2.1 Data Preparation

The dataset consists of 10,000 samples generated using the dataset created in Chapter 3, where each sample is a character sequence padded to a fixed length. The data was split into training and validation sets with 9,000 and 1,000 samples respectively.

4.2.2 Loss Function and Optimization

The model was trained using the Cross Entropy Loss function. A critical detail in the training process is handling the padding tokens. The loss function was configured to ignore those padding tokens to ensure that the model is not penalized for predicting padding characters, allowing it to focus solely on the meaningful arithmetic sequence.

- **Optimizer:** Adam with a learning rate of 0.001.
- **Loss:** `nn.CrossEntropyLoss(ignore_index=13)`.

4.2.3 Teacher Forcing

During training, we utilized “teacher forcing.” The model receives the entire ground-truth sequence as input and predicts the next token for every position. The predictions are compared against the target labels, which are the input sequences shifted by one position.

4.2.4 Training Results

The model was trained for 150 epochs. The loss curves indicate stable convergence, with the training loss decreasing from an initial 1.70 to approximately 0.83, and validation loss stabilizing around 0.87, as shown on figure 4.1.

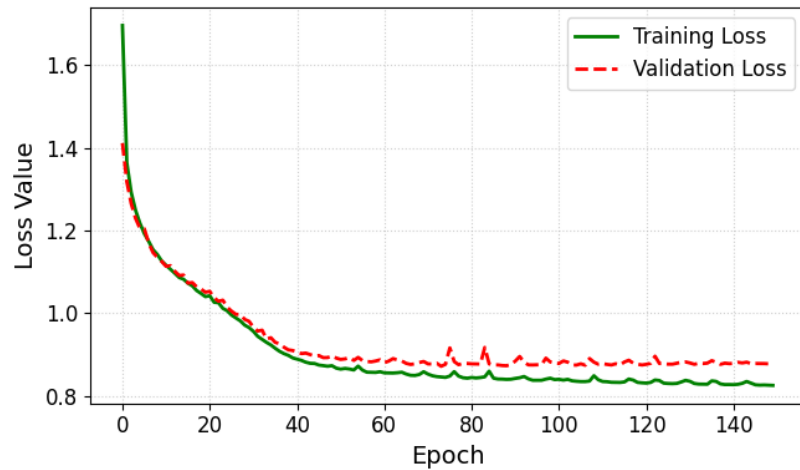


Figure 4.1: Training and Validation Loss over 150 Epochs

4.3 Evaluation

The model was evaluated on a test set of generated addition problems.

- **Overall Accuracy:** 97.21% (percentage of equations where the entire answer string was correct).
- **Token Accuracy:** 98.48% (percentage of individual tokens predicted correctly).

4.3.1 Error Analysis

To generate answers for new addition problems, we implemented an autoregressive prediction loop. Unlike training, where the full sequence is known, prediction generates the answer one character at a time and then analysed the predictions.

- **Single Digit Answers:** High error rate (57.63%). This is likely due to the small sample size of single-digit answers available in the dataset (only 59 samples) compared to the thousands of double and triple-digit examples.
- **Double Digit Answers:** Low error rate (4.46%).
- **Triple Digit Answers:** Very low error rate (0.43%).

The results demonstrate that the RNN successfully learned the algorithmic pattern of addition for standard double and triple-digit sums, treating the operation as a sequence prediction task.

Chapter

5

Conclusions

5.1 Technologies Used

To accomplish this work, it was necessary to use several external libraries beyond the standard Python library: *TensorFlow*, was used in a first instance, but generated way to many incompatibilities and required some adaptations to existing functions, what was revealed confusing and subsequently discarded. *PyTorch*, used for building, training, and evaluating the deep learning models, *NumPy*, used for numerical operations and efficient array manipulation, *Matplotlib*, used for visualizing training loss and learning curves, and *scikit-learn*, used for calculating evaluation metrics.

5.2 Conclusion

In this project, we successfully implemented and evaluated diverse deep learning architectures for sequence processing. In Task 1, we benchmarked LSTM networks against pre-trained Transformer encoders for POS tagging, observing the superior performance of attention-based models. In Task 2, we explored the generative capabilities of decoder-only Transformers (GPT) for arithmetic tasks. Finally, in Task 3, we implemented a recurrent solution using a custom RNN with teacher forcing and autoregressive generation. Through these tasks, we deepened our understanding of the fundamental shift from recurrent to attention-based mechanisms in modern artificial intelligence development.

