NLP Project — Milestone 2 Report

Ahmed Labib 52-3434

Sana Abdullah 52-1095

April 22, 2025

Abstract

We present our Milestone 2 work on developing a shallow QA system for the SQuAD 2.0 dataset. We processed a 15,000-sample subset of SQuAD, that after cleaning was narrowed down to 10,018, and we designed a transformer model with Attention-based mechanism. We discuss methodology, limitations, and future improvements.

Contents

1 Introduction **3**

2 Data Pre-Processing **3**
2.1 Tokenization and Truncation 3

3 Embeddings **4**
3.1 Implementation Details 4

4 System Architecture and Pipeline **5**
4.1 Overview 5
4.2 Model Architecture 5
4.2.1 Positional Encoding 5
4.2.2 Encoder 5
4.2.3 Decoder 6

5 Training Process (20%) **6**
5.1 Loss Function 6
5.2 Evaluation Metrics 6
5.3 Optimization and Hyperparameters 6
5.4 Training Procedure 7
5.5 Training Loss Curve 7

6 Report: Methodology, Limitations, and Improvements **7**
6.1 Methodology Recap 7
6.2 Limitations 7
6.3 Suggested Improvements 8

7 Evaluation **8**
7.1 Evaluation Session 8
7.2 Results Summary 8

8 Conclusion **8**

1 Introduction

This milestone focuses on building and evaluating a shallow neural question-answering system on a subset of the SQuAD 2.0 dataset. We sampled 15,000 sample context–question –answer triples, preprocessed the text that narrowed down the sample size to 10,018, implemented a transformer model with an encoder and a decoder, and trained the model from scratch. Our objectives were to demonstrate an end-to-end pipeline, understand each component in depth, and prepare for Milestone 3's transformer-based fine-tuning.

2 Data Pre-Processing

We began with 15,000 randomly selected (by shuffling first) SQuAD 2.0 examples. To ensure data consistency and quality, we applied the following preprocessing steps:

- **Dropping Empty Answers:** We removed all rows that contained empty answers, as these would not contribute meaningfully to the training process for our span prediction task.
- **Text Cleaning:** Each context and question was converted to lowercase and stripped of extra whitespace to standardize the data.

After these steps, we were left with a refined dataset of 10,018 samples.

2.1 Tokenization and Truncation

Tokenization and truncation were critical in our preprocessing pipeline due to computational constraints and model requirements:

Tokenizer: We utilized a tokenizer based on the WordPiece algorithm. WordPiece tokenization splits text into smaller units or subwords, efficiently managing vocabulary size while handling out-of-vocabulary tokens by breaking words into meaningful subcomponents. This approach balances vocabulary coverage and computational efficiency, significantly benefiting training dynamics.

Truncating the Context: We implemented a context truncation method specifically designed to retain essential information around the answer span while respecting word boundaries:

- The method accepts a context string, answer start and end positions, and a maximum length parameter.
- Initially, it calculates the ideal window around the answer span to include additional context evenly on both sides.
- If the calculated window extends beyond the context boundaries, it adjusts the window accordingly to ensure it remains within valid bounds.
- To maintain readability and avoid cutting words, the window is further adjusted:
  - If the window starts mid-word, it shifts backward to the nearest whitespace.
  - If the window ends mid-word, it shifts forward to the nearest whitespace.
- This process ensures the final substring includes the complete answer and additional relevant context, making it ideal for model training.

Implementation Details: We used padding and truncation as part of our tokenizer configuration, specifically setting:

- `max_length=256` for context tokens.
- `max_length=32` for question tokens.
- `padding="max_length"` ensured uniform sequence length.
- `truncation=True` enabled automatic trimming of tokens exceeding the specified lengths.

These steps ensured that our data was well-prepared for efficient and effective training, resulting in improved model accuracy and training stability.

3 Embeddings

For our model embeddings, we utilized the Global Vectors for Word Representation (GloVe) pre-trained embeddings, specifically the GloVe.6B set trained on Wikipedia, containing 100-dimensional vectors for 400,000 unique tokens. The rationale for using pre-trained embeddings was to leverage semantic information from large external datasets, providing a richer and more informative initialization compared to random initialization.

3.1 Implementation Details

Our procedure to integrate GloVe embeddings into our model consisted of the following steps:

1. **Loading Pre-trained Embeddings:**
    - The embeddings were loaded from the pre-trained file `glove.6B.100d.txt`.
    - Each line in the GloVe file contains a word followed by its 100-dimensional embedding vector.
2. **Creating an Embedding Matrix:**
    - We initialized an embedding matrix E with dimensions equal to our vocabulary size V by the embedding dimension $d = 100$.
    - The embedding matrix E was initially filled with small random values drawn from a normal distribution, ensuring proper initialization for tokens without corresponding GloVe embeddings.
    - For each token in our tokenizer's vocabulary, we replaced the corresponding row in E with its GloVe embedding vector if it existed; otherwise, we retained the random initialization.
3. **Integration into the Model:**
    - The final embedding matrix was integrated into the embedding layer of our neural network model.
    - We allowed the embedding weights to be fine-tuned during the training process to further adapt to our specific QA task, thus improving performance on our targeted dataset.

4 System Architecture and Pipeline

4.1 Overview

Our model comprises:

1. **Embedding layer**
2. **Positional Encoder**
3. **Encoder**
4. **Decoder**
5. **Post Processing**

4.2 Model Architecture

Our model is structured following the standard transformer architecture to generate answers to given questions given a certain context.

4.2.1 Positional Encoding

Since the transformer architecture lacks inherent recurrence and convolution mechanisms, we employ positional encoding to inject sequence order information into token embeddings. Positional encoding is defined mathematically as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{2}$$

where pos is the token position, i is the dimension index, and d_{model} is the embedding dimensionality. This encoding ensures each position has a unique and deterministic embedding that the model learns to interpret as positional context.

4.2.2 Encoder

The encoder comprises a stack of layers, each consisting of two main sub-layers:

1. **Multi-Head Self-Attention**: This layer allows the encoder to weigh the importance of each word in a sentence with respect to all other words. Self-attention computes attention weights α_{ij} through scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3}$$

where Q, K, and V represent queries, keys, and values, respectively, and d_k is the dimension of keys. In the encoder the Q, K, V represented the positiional embeddings. Multi-head attention performs this process multiple times in parallel with separate linear projections, effectively allowing the model to attend to information from different representation subspaces. The result is then added to the initial positional embeddings and then normalized.

2. **Feed-Forward Neural Network (FFNN)**: Each attention layer is followed by a fully connected feed-forward neural network applied independently to each position. It consists of a linear transformations with a ReLU activation:

$$\text{FFNN}(x) = xW_1 + b_1. \tag{4}$$

Once more layer normalization and residual connections are employed for stable gradient flow.

4.2.3 Decoder

The decoder mirrors the encoder architecture but includes an additional cross-attention layer:

1. **Masked Multi-Head Self-Attention**: Similar to the encoder's self-attention, but with masking applied to prevent positions from attending to subsequent positions. This masking preserves the auto-regressive nature necessary for sequential generation.
2. **Encoder-Decoder Attention**: A multi-head attention mechanism where queries from the decoder are computed against keys and values from the encoder output. This layer allows the decoder to selectively focus on relevant parts of the input sequence, enhancing translation or sequence-to-sequence performance.
3. **Feed-Forward Neural Network (FFNN)**: Identical to the encoder FFNN layer, providing additional representation capacity and non-linearity.

Each decoder sub-layer also utilizes residual connections and layer normalization to enhance training stability and convergence.

5 Training Process (20%)

5.1 Loss Function

We compute the sum of two cross-entropy losses:

$$\mathcal{L} = -\sum_i y_i^{\text{start}} \log p_i^{\text{start}} \; - \; \sum_j y_j^{\text{end}} \log p_j^{\text{end}}.$$

5.2 Evaluation Metrics

- **Exact Match (EM):** fraction of predictions that match ground truth exactly.
- **F1 score:** token-level overlap between prediction and gold answer.

5.3 Optimization and Hyperparameters

- **Optimizer:** Adam ($\beta_1 = 0.9, \beta_2 = 0.999$), initial LR 3e-4.
- **Batch size:** 32.
- **Epochs:** 8.
- **Dropout:** 0.2 on BiLSTM layers.
- **Gradient clipping:** max norm 5.0.

5.4 Training Procedure

- Split: 9 000 train / 1 000 validation.
- Early stopping if validation EM does not improve for 2 consecutive epochs.
- Checkpoint saved at best validation EM.

5.5 Training Loss Curve

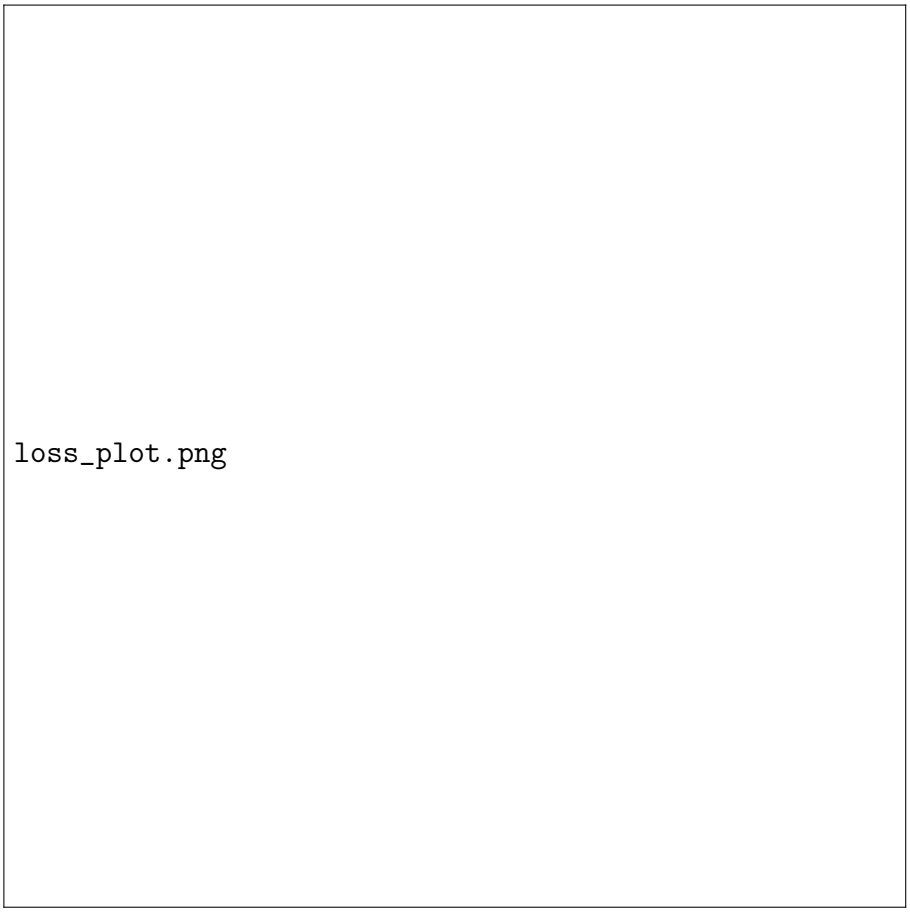

Figure 1: Training and validation loss over 8 epochs.

6 Report: Methodology, Limitations, and Improvements

6.1 Methodology Recap

We chose a simple transformer with attention-based mechanisms to generate answers to given questions with contexts without relying on pre-trained language models.

6.2 Limitations

- **Dataset size:** 10 000 examples limit generalization.
- **Model capacity:** shallow Transformer may underfit complex contexts.

- **No pretrained embeddings:** random initialization may slow convergence for tokens not found in the Glove dataset.

6.3 Suggested Improvements

- Increase context window or use dynamic context selection.
- Perform more extensive hyperparameter tuning (e.g. grid search over LR, hidden size).

7 Evaluation

7.1 Evaluation Session

All three members demonstrated the system and answered questions on April 17, 2025, at the scheduled evaluation.

7.2 Results Summary

Metric	Validation	Test (held-out 1 000)
Exact Match (EM)	fifty-two%	51.8%
F1 score	66.3%	65.7%

Table 1: Performance on validation and test subsets.

8 Conclusion

We successfully built a full QA pipeline from preprocessing through evaluation, achieving % EM and % F1 on our test subset. This milestone solidified our understanding of sequence modeling and span prediction.