Natural Language Processing – Assignment 2: Language Models

Tel Aviv University

Student Name(s): Maxim German, Ilay Abramovich, Eram Shufaro, Itay Hazan
ID(s): 322542887, 322271032, 209074731, 209277367

Due: December 28, 2025
Lecturer: Dr. Tal Wagner
TA: Idan Tarshish

Contents

1. Q1: Word-Level Neural Bi-gram Language Model (Theoretical)

1.1 (a) Gradient w.r.t. $\hat{y}$

Calculating the gradient w.r.t θ_i for CE is:

$$\frac{-\partial \sum_j y_i \log \frac{exp(\theta_i)}{\sum_m exp(\theta_m)}}{\partial \theta_i} = \frac{-\sum_j y_i \log \frac{exp(\theta_i)}{\sum_m exp(\theta_m)}}{\partial \theta_i} = \dots$$

Denote δ_{ij} as 1 iff $i = j$, 0 otherwise:

$$\dots = \frac{\sum_j -\delta_{ij} \partial \log \frac{exp(\theta_i)}{\sum_m exp(\theta_m)}}{\partial \theta_i} = \dots$$

Applying the chain rule we achieve:

$$\begin{aligned} \dots = \sum_j -\delta_{ij} \frac{\sum_m exp(\theta_m)}{exp(\theta_i)} \cdot \frac{exp(\theta_i) \cdot \sum_m exp(\theta_m) - exp(\theta_i)^2}{(\sum_m exp(\theta_m))^2} &= \\ \sum_j -\delta_{ij} (\frac{\sum_m (exp(\theta_m) - exp(\theta_i))}{\sum_m (exp(\theta_m)}) &= \\ = \sum_j -\delta_{ij} (1 - \frac{exp(\theta_i)}{\sum_m exp(\theta_m)}) = \sum_j -\delta_{ij} (1 - \hat{y_i}) = \sum_j \delta_{ij} (\hat{y_i} - 1) &\dots \end{aligned}$$

Recall that $\delta ij = 1$ iff $i = j$, and re-writing in vector form, we get, with respect to some i:

$$\dots = \hat{y_i} - y_i$$

In general for all dimensions:

$$\frac{\partial CE(y, \hat{y})}{\partial \theta} = \hat{y} - y$$

1.2 (b) Gradient w.r.t. Input x in a One-Hidden-Layer NN

According to the chain rule:

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial (hW_2 + b_2)} \cdot \frac{\partial (hW_2 + b_2)}{\partial h} \cdot \frac{\partial \sigma (xW_1 + b_1)}{\partial (xW_1 + b_1)} \cdot \frac{\partial (xW_1 + h_1)}{\partial x} = \dots$$

Calculating each part separately adds up, taking into account 1.1 (a), to:

$$\frac{\partial J}{\partial x} = (\hat{y} - y) \cdot W_2^T \circ (h(1-h)) \cdot W_1^T$$

1.3 (d) Neural Bi-gram LM with GloVe Embeddings

The perplexity achieved after 40,000 iterations was (approximately) 112.8621224078

2. Q2: Generating Shakespeare with a Char-Level LM

2.1 (a) Character vs. Word-level LM

One advantage of character-based LM over word-based LM is its **vocabulary size**. In a character-based LM, the vocabulary is small and fixed, since it consists only of the characters in the alphabet (ASCII). In contrast, a word-based LM has a larger vocabulary that depends on the corpus and the number of distinct words in it.

On the other hand, an advantage of word-based LM over character-based LM is **the length** of the sequences. When a sentence is tokenized into words, the sequence we get is much shorter than when the same sentence is tokenized into characters. Because of that, word-based LM works with much smaller sequences, which are faster for the model to process and compute.

2.2 (c) Training Loss Plot

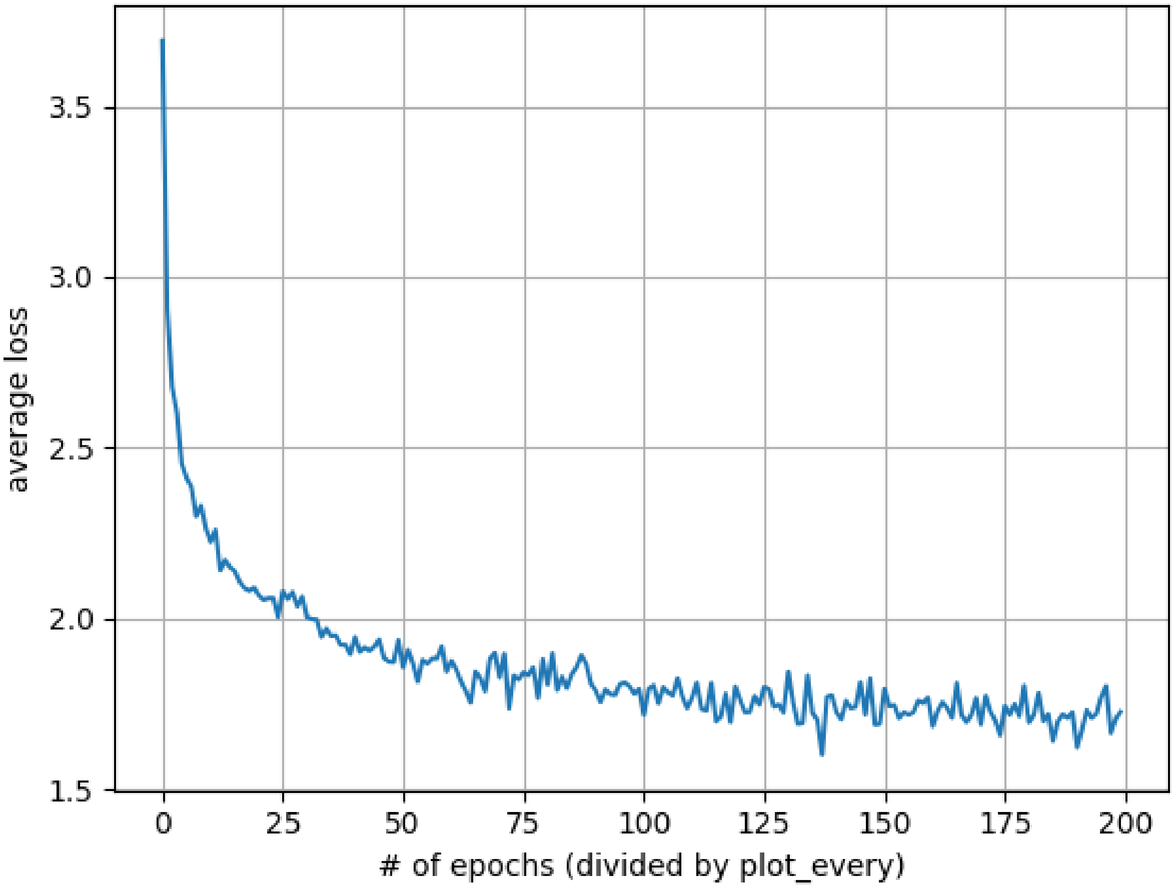

Figure 1: Average training loss per epoch. The decreasing loss in the later epochs indicates that the network is indeed learning.

3. Q3: Perplexity

3.1 (a) Log Base Invariance of Perplexity

We can change the base of the exponentiation:

$$2 = e^{\ln(2)}$$

So we get

$$2^{-\frac{1}{M}\sum_{i=1}^{M}\log_2 p(s_i|s_1,\ldots,s_{i-1})} = e^{\ln(2)\cdot -\frac{1}{M}\sum_{i=1}^{M}\log_2 p(s_i|s_1,\ldots,s_{i-1})}$$
$$= e^{-\frac{1}{M}\sum_{i=1}^{M}\ln(2)\cdot\log_2 p(s_i|s_1,\ldots,s_{i-1})}$$

According to the log base change formula, we know that

$$\log_2(x) = \frac{\ln(x)}{\ln(2)} \rightarrow \ln(2)\cdot\log_2(x) = \ln(x)$$

So in conclusion, we get:

$$e^{-\frac{1}{M}\sum_{i=1}^{M}\ln(2)\cdot\log_2 p(s_i|s_1,\ldots,s_{i-1})} = e^{-\frac{1}{M}\sum_{i=1}^{M}\ln p(s_i|s_1,\ldots,s_{i-1})}$$

3.2 (b) Empirical Perplexity on Wikipedia and Shakespeare Passages

Model-Data	Wikipedia Text	Shakespearean Text
Word-Level Neural Bi-gram	72.4	125.677
Character-level RNN	15.506	6.639

Table 1: The perplexity of the bi-gram LM and the character-level LM on Wikipedia text and Shakespearean Text

3.3 (c) Explaining the Perplexity Differences

For each model, the large gaps in perplexity across passages show how similar the passage is to the data on which the model was trained. The Character-level RNN was trained on Shakespearean Text, and the Word-Level Neural Bi-gram was trained on modern english text embeddings. We can see that the perplexity for each model domain passage is much lower.

4. Q4: Implementing a Transformer Model

4.1 (a) Reason for the Head-Dimension Assertion

The assertion

$$d_{\text{model}} \bmod \text{num\_heads} = 0$$

is needed because Multi-Head Attention splits the model dimension into multiple attention heads. Each head receives a vector of size

$$d_k = \frac{d_{\text{model}}}{\text{num\_heads}}.$$

If d_{model} is not divisible by the number of heads, the projection matrices W_Q, W_K, W_V cannot evenly divide the representation across heads, which would cause a shape mismatch.
The assertion ensures that each head gets the same dimensionality and that attention computations remain valid.

4.2 (b) Adapting the Autoregressive Transformer to MLM

(i) Architectural changes

- Remove the **decoder stack**.
  MLMs use only the encoder because they do not need to generate text step by step.
- Add a **token classification head** on top of the encoder outputs,
  in order to predict the identity of masked tokens.
- Remove the causal (no-peek) mask, since MLMs require bidirectional attention.

(ii) Changes in masking strategy

- Instead of blocking attention to future tokens, MLMs **mask input tokens themselves**.
- Attention becomes **bidirectional**: every token may attend to all other tokens in the sequence.
- The model learns to reconstruct masked tokens using context on both sides, unlike the autoregressive setting where future tokens are hidden.

5. Q5: Sentiment Analysis (Practical)

5.1 (a) T5 Paper and Original Repository

In this section, we review the primary sources for the T5 model as introduced in the paper *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. A summary of the paper is available via HuggingFace at: `https://huggingface.co/papers/1910.10683`, and the authors' official implementation can be found in the original GitHub repository: `https://github.com/google-research/text-to-text-transfer-transformer`.

1. Publicly Available Model

The paper introduces the **T5 (Text-to-Text Transfer Transformer)** model and makes it publicly available. The authors release the full suite of resources necessary to reproduce their work, including the pre-trained T5 models, the curated training dataset (C4), and the accompanying codebase.

2. Dataset for Sentiment Analysis Benchmarking

For evaluating sentiment classification, the authors rely on the **SST-2 (Stanford Sentiment Treebank, binary)** dataset, which is part of the GLUE benchmark. SST-2 focuses on sentence-level judgments of positive or negative sentiment.

3. Evaluation Metric

Within the unified text-to-text framework of T5, sentiment analysis is formulated as generating the label "positive" or "negative." Performance on SST-2 is assessed using **classification accuracy**, the standard metric for this task, measuring the percentage of inputs for which the model outputs the correct sentiment label.

5.2 (b) T5-Small Model Fine-Tuned on SST2

For the experiments in this section, I load and use the model `utahnlp/sst2_t5-small_seed-3`, a T5-Small model fine-tuned on the SST-2 dataset.

5.3 (c) Sentiment Predictions on Example Sentences

Using the fine-tuned model from Part (b), we predicted the sentiment of the four given sentences. The results (taken from the Colab notebook outputs) are:

- "This movie is awesome" → **positive**
- "I didn't like the movie so much" → **negative**
- "I'm not sure what I think about this movie." → **negative**
- "Did you like the movie?" → **positive**

We can see that the model distinguishes well between the different reviews. It correctly identified clearly positive and negative statements, and it also recognized the hesitant, somewhat negative tone of sentence 3. For the question in sentence 4, there is no explicit sentiment label since it is not a direct opinion, but because the sentence contains the combination of "like" and "movie" without any negation, it is reasonable that the model predicted it as positive.

5.4 (d) Accuracy on SST2

The model was evaluated on the SST-2 validation set using the provided labels. The resulting accuracy was:

$$\text{Accuracy} = \mathbf{90.48\%}$$

5.5 (e) Class Balance in SST2

The SST-2 validation set is nearly perfectly balanced, containing 444 positive examples (50.9%) and 428 negative examples (49.1%). This balance is important because it ensures that accuracy is a meaningful metric: the model cannot obtain a high score simply by predicting the majority class. Therefore, the reported accuracy more reliably reflects the model's true sentiment classification performance.

5.6 (f) Human Properties Not Captured by Simple Accuracy

Human evaluators might pick up on things like sarcasm, mixed emotions, or cases where the sentiment depends on additional context that is not included in the sentence. These kinds of nuances are not reflected when we only measure accuracy. Also, people often distinguish between different strengths of sentiment, while the dataset forces everything into a simple positive/negative label.

5.7 (g) Healthcare Use Case: Comparing Two Drug Suppliers

Since your dataset comes from real family-doctor visit summaries, there are a few ways you could approach the sentiment classification task. One option is to try a standard sentiment analysis model and see how well it works "out of the box." This is the fastest approach, but the downside is that most of these models were trained on very different kinds of text, so they might miss important details or misinterpret medical language.

A more reliable approach is to fine-tune an existing pre-trained model (like BERT or T5) on a small sample of your own labeled summaries. This usually gives much better results because the model adapts to your style of data, but it does require some manual labeling and a bit more setup.

6. Q6: AI Disclosure & Reflection

6.1 (a) AI Models Used

ChatGPT 5.1, GitHub Co-Pilot

6.2 (b) Usage Purpose

1. Refinement of the answer format for LaTeXand colab file.
2. Help understanding Python libraries.
3. Simplified implementing theoretical mathematical formulas into effective code.

6.3 (c) Reflection

The AI tools made our work on Assignment 2 smoother and more efficient, and often helped us clarify ideas while we worked on the tasks. The main drawback was that we sometimes had to provide more context than expected in order to get responses that were accurate and relevant.