Introduction to Computational Linguistics

Contest 2: Predictive typing

Due date: 19 Oct 2025 (no late submission allowed)

Task Overview

Your goal is to train (or use) a language model to predict the next word in a sequence. We will simulate a predictive keyboard, like those found on mobile phones or tablets. Unlike real keyboards, however, this version will not include spell check.

How you will be graded

This assignment may be done individually or in pairs. Late submissions are not allowed because everyone's accuracy results will be combined together. Your assignment will be evaluated based on two equally important criteria:

- Accuracy on the test set (50%)
- Report (50%) same criteria as Contest 1
    - Process and analysis (30%): The quality of your approach and insights, documented in your written report. The report should demonstrate an understanding of the principles for improving model accuracy and the factors affecting model accuracy.
    - Quality of writing (20%): must be easy to read and clear. The tables and figures must be well-formatted.

You can use any approach for your final system, but at minimum you are required to implement two of these baseline models:

1. Trigram model
2. 4-gram model
3. RNN model (e.g. vanilla RNN, GRU, or LSTM)

Provided materials:

1. Training set (train.zip): This file contains approximately 128 million tokens. The text has already been lowercased and tokenized according to the Penn Treebank conventions. In this format, punctuation marks and possessive markers are treated as separate tokens, and parentheses are replaced with bracket symbols. For example:

I'm going to the store (tomorrow)
→ i 'm going to the store - lrb - tomorrow - rrb -

Students may also make use of additional training data if desired. However, any supplementary data must be tokenized in a way that is as consistent as possible with the format of the provided training set.

2. dev_set.csv: This dataset will be used to evaluate model performance during development. Each row contains a context, the first letter of the true answer, and the full correct answer. For example:

context	first letter	answer
the zambian government and the african development bank - lrb - adb - rrb - will be financing a poverty alleviation project to promote income generation and	e	employment

To make this task easier, we will provide the first letter of the true answer. For example, if the correct answer is *employment*, the column first letter will contain the letter *e* as a hint—similar to how a user types the first letter before completing the whole word.

3. test_set_no_answer.csv: This dataset has the same format as the development set, except the *answer* column is omitted. Students must run their trained language model to predict the missing word based on the provided *context* and *first letter*. The predictions will form the basis of the final evaluation.

4. check.py: This script ensures that the submitted prediction file is in the correct format. Specifically, it checks that the number of predictions exactly matches the number of rows in the test set. Students should run this script to confirm their submission before uploading.

What to submit:

1. **test_set_pred.txt** must contain the predictions for the test set. Each row should contain only one word. For example, row 1 should contain the prediction for row 1 of *test_set_no_answer.csv*:

```
day
the
morning
was
…
```

You should run

```
python check.py
```

and make sure it passes before submitting on MCV.

2. **report.pdf :**Your written report should be no longer than five pages and should present a clear account of your experiments and your analysis of the results. Think of it as a research-style paper: the goal is not only to show what you did, but also to demonstrate that you understood the task, designed the experiments carefully, and interpreted the outcomes critically.

   Note on the use of AI: You are welcome to use an AI assistant of your choice to help you write up the report. But you must make sure that the model details, figures, and numbers are accurate and be sure to form your own idea of what you observe in the experiment and what you personally learn from this assignment in the conclusion.

Data

Explain clearly what data you used for training and evaluation and in what amount. How do you make a decision on how much data to use and how do you sample them? Where do you source your data?

Model

Discuss all of the models you attempted, not just the final one. For each, explain the motivation for trying it (for example: n-gram language model, recurrent neural network, transformer-based model). You should also explain which hyperparameters you tuned (for example: learning rate, embedding size, number of layers, dropout, context window size) and how you decided on their values.

Results

Provide a comparison of the results in the form of a table that clearly shows how the models differ. The table should include performance metrics (e.g., accuracy or optionally top-k prediction accuracy), and you may add graphs if they clarify trends. Be sure to highlight not just the raw numbers, but also the reasons behind the differences. This discussion should make it clear to the reader which factors contributed to better performance and which were less important

Conclusion

Summarize your findings by stating which model you ultimately chose for the test set and why. Your reasoning should connect back to the evidence from the results section. Note that your final model may be trained on the full training set—make sure to explain if you retrained it this way. End your report by briefly reflecting on what worked well, what challenges you faced, and what you might try differently if given more time or resources..

Getting Started

The goal of this assignment is to build a predictive model for next-word prediction. Unlike full language modeling, you do not need to compute probabilities for all candidate words. Instead, your model should simply output the most likely next word given a context and the first letter of the target word.

A recommended baseline is a trigram model. Begin by training it on a small dataset (e.g., 100 sentences) to confirm your implementation works before scaling up or you will unnecessarily waste a lot of training time on a wrong implementation.

Example pseudocode for training a trigram model:

```
    for i in range(len(tokens)- 3 - 1):
      context = tuple(tokens[i:(i+3-1)])
      word = tokens[(i+3-1)]
      if context not in counter:
        counter[context] = Counter()
      counter[context][word] += 1
```

For prediction, use `counter[trigram_context].most_common()` and select a word beginning with the given first letter. If the trigram context is not found in the counter, return `'  '`.

You will need a function to compute accuracy on the development set.
(Tip: You can prompt AI to "Write a Python function that computes accuracy from two lists").
On the development set, you should expect results in the following range:

Model	Training data	Dev accuracy
Trigram	10,000 sentences	0.20
Trigram	100,000 sentences	0.35

Further suggestions

- **Backoff methods:** If a trigram context is missing, fall back to a bigram or unigram.
- **Higher-order models:** Extending to 4-gram or 5-gram models only requires changing the size of the context tuple.
- **KenLM with Kneser–Ney smoothing (as practiced in class):** Train a language model and compute sentence probabilities after inserting each candidate word. Restrict predictions to words from the training vocabulary.

  Example pseudocode:

```
    for word in vocab:
     if word[0] == first_letter:
       sentence = context + ' ' + word
```

```
        prob = kenlm.compute_probability(sentence)
        if prob > max_prob_so_far:
          best_word = word
    return best_word
```

- RNNs: Training a recurrent neural network can take hours, so you should only attempt this after experimenting with at least trigram models. If you try this approach, implement a word-level vanilla RNN and enable GPU acceleration on Colab.
- Pre-trained models: GPT-1 or GPT-2 can provide strong performance and serve as competitive baselines.

Additional Data Sources (optional)

If you wish to explore beyond the provided training set, you may use additional corpora although we have already provided you a copious amount of data. Ensure the tokenization is consistent with the provided data.

- Gigaword corpus (New York Times) (2.37Gb) https://drive.google.com/file/d/13B5RDlDOHllTjFnRcj671yviKQ80Rjgi/view?usp=sharing
- Wikipedia text dump (5.34Gb) http://kopiwiki.dsd.sztaki.hu/
- Twitter data of various groups and sizes https://data.world/datasets/twitter
- Original data is drawn from https://huggingface.co/datasets/gigaword