CompLING Practice 1: Next Word Distribution

Freda Shi

September 1, 2025

Please note that this task is deprecated. Any submission after Sept 06 should work on other tasks.

This practice task is part of the practice task set for prospective interns and research assistants at the Computational Linguistics Lab at the University of Waterloo. See also the GitHub repository for coding style and submission instructions.

All the computing for this task can be done on Google Colab (free version) with or without a GPU or a reasonable laptop. I recommend using or at least trying a GPU, even if it is not necessary for this task. This will help you gain experience with using GPUs for large models.

1 Introduction and Background

This task will familiarize you with pre-trained **autoregressive** language models, backpropagation, and the PyTorch and Hugging Face Transformer libraries.

1.1 Autoregressive Language Models

An autoregressive language model predicts the next word in a sequence given the previous words. In probabilistic terms, the model defines a probability distribution over sequences of words (e.g., sentences or paragraphs) by factorizing the joint probability of the sequence into a product of conditional probabilities:

$$P_\Theta(w_1, w_2, \ldots, w_n) = P_\Theta(w_1) \cdot \prod_{i=2}^{n} P_\Theta(w_i \mid w_1, \ldots, w_{i-1}),$$

where w_i is the i-th word in the sequence, and Θ refers to model parameters.

You might have heard of Chat-GPT or GPT-4. The core idea is quite simple: given a sequence of words (more precisely, subwords), the model predicts the next subword. Here, we may view the GPT model parameters as Θ in the above equation.

In this task, you will work with pretrained GPT-2 models using the HuggingFace Transformers library. Specifically, you will use `gpt2` and `gpt2-large` for all your experiments. Please check carefully the documentation of the Hugging Face Transformers library to understand how to load the models and use them for prediction: `https://huggingface.co/openai-community/gpt2`. Hint: for this task, you may wish to check the `GPT2LMHeadModel`.

Technical detail note. Please pay special attention to the bos_token, eos_token, and pad_token in the tokenizer and understand when and how to use them. Different model families have different conventions for these tokens—you are highly encouraged to check out other models, such as BERT, Llama, and OPT; you should always check the documentation or run the tokenizer code before experiments to ensure correct usage.

1.2 Backpropagation

Backpropagation is a method used in artificial neural networks, among many other machine learning models, to calculate the gradient of the loss function with respect to the model parameters. The idea is quite simple and intuitive: assume we have a convex loss function $L(\Theta, \mathcal{D})$, where Θ is the model parameters and $\mathcal{D}$ is the data, and we want to minimize the loss function; what we can do is to move a small step in the opposite direction of the gradient of the loss function:

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} L(\Theta, \mathcal{D}),$$

where η is the learning rate. In actual practice, the loss function is usually, or almost always, continuous but is not always, or almost never, convex. Still, the idea generally works well—according to the definition of gradient, if we set the learning rate to be low enough, the value of the loss function will decrease over time. It might stop at a local minimum, but it is usually good enough for most practical purposes.

1.3 Transformers and Transformer Language Models

Please check out the Transformers paper (Vaswani et al., 2017) and the GPT-2 paper (Radford et al., 2019). You are also highly encouraged to check out the source code of the

GPT-2 model in the Hugging Face Transformers library.

2 Task Description

First, please try loading the GPT-2 models (`gpt2, gpt2-large`) from the Hugging Face Transformers library and familiarize yourself with the model inference process. For each task, please report the results for both models.

Your code should follow the Google style guide, excluding the rules E501 (maximum 80 characters per line—the CL Lab uses a soft rule here; however, please ensure the line is not too long to affect understanding) and E731 (do not assign a lambda expression). See the GitHub repo for more details regarding pre-commit checks.

2.1 Next Word Prediction

Task 1: Write down the ten most probable next tokens, ordered by probability from high to low, given the following input text: "*Ludwig the cat*" (only the three words, and not including the quotes). What is the probability of the most probable next token? Please report the results for both models in your report, and submit your code as `prediction.py`.

2.2 Model Fine-Tuning

In this part, you will use the backpropagation algorithm to manipulate the parameters. This approach is also called fine-tuning the model: starting from a pre-trained model, you modify the model parameters to make the model better fit the data you have.

Task 2: Design a loss function, and modify the model parameters by minimizing the loss with backpropagation to make sure that the model (almost) always predicts the word "*stretches*" after seeing "*Ludwig the cat*".

- Describe what you did in your report, including the information below:
    1. The formulation of your loss function.
    2. The optimizer and learning rate you used.
    3. How you modified the model parameters.
    4. How the loss changed before and after your operation.
    5. The final loss value.

6. The probability of predicting the word "*stretches*" given "*Ludwig the cat*" before and after your operation.

- Submit your code as `finetuning.py`.

Feel free to use any optimizer and learning rate as long as it works as desired. If you have learned the term overfitting, do not worry about it.

2.3 Model Fine-Tuning with Frozen Parameters

Now, you may only modify the parameters of the used noncontextualized subword embeddings by "*Ludwig the cat*". That is, the text "*Ludwig the cat*" will be tokenized into a few subwords by the default GPT-2 tokenizer, and you may only modify the corresponding noncontextualized subword embeddings. Do not modify the position embeddings or the other parts of the model. Again, do not worry about overfitting.

Hint: you need to identify which part of the model weights refers to the noncontextualized subword embeddings. Looking at the shape of the weights in each module will be very helpful.

Task 3: What are the subwords of "*Ludwig the cat*"? Write down the subwords in your report.

Task 4: Your goal is the same as in Task 2, but you are only allowed to tune the noncontextualized subword embeddings used by "*Ludwig the cat*".

- Describe what you did in your report, with the same requirements as in Task 2.
- Submit your code as `prompt.py`.

Submission Instruction

Please use the Google Form to submit your report and code.

Your report should be in PDF format. Feel free to reuse this document and fill in the answers. The LaTeX source code is provided in the GitHub repository.

Your code should be written in Python and should be runnable. Submit your code as a `.zip` file. **In your `.zip` file, please only include `.py` files and a `README.md` file that explains how to run your code.** List the essential dependencies (e.g., Hugging Face versions) in the README file. Including any Python or LaTeX auxiliary files will result in a desk rejection of your application.

References

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. 2

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*. 2