The best permutation by perplexity

Introduction

Permutation tasks in NLP - the rearrangement of sequences or tokens for better reading this text. It can be used for:

1. Grammarly correct document summarization.
2. Rearranging paragraphs in a shuffled document, such as news articles or reports, to improve readability and logical flow.
3. Improving Search and Recommendation Systems by permuting search keywords to identify the best query formulation for retrieving accurate results.
4. Rearranging responses in a conversation in chatbots to maintain context and logical flow in customer support or virtual assistants.
5. In Language Learning Applications identifying incorrect permutations of words that disrupt grammatical correctness.
6. Aligning and reordering content in translations documents where sentence order differs between languages, such as aligning English and Japanese subtitles.
   And more other tasks ...

As example of permutation task I take kaggle competition https://www.kaggle.com/competitions/santa-2024/overview. In this competition you have samples with Christmas stories that have words in wrong order, and you need to rearrange this words to get proper story back. I will try to solve this puzzle using approaches that was used for permutation tasks, which no one used in the public kaggle notebooks, but I can find this in the articles. Spoiler: it is IBIS approach (improvements for beam search and simulated annealing approaches) that can find the good permutations for faster time, than other available approaches.

Team

This project was created by Iuliia Fokina

Related Work

This section I divide on two parts: related articles and related kaggle notebooks.
Related articles:

- IBIS approach (iterative shuffling) [Malkin et al., 2021]. IBIS iteratively infers the word order that has the lowest negative log-likelihood under GPT-2. At each step, the sentence is cut into pieces, which are then rearranged. After several such k-opt moves, the original order is found. They improved beam search approach inability to plan. It looks similar as our simulated approach in baselines. The main difference

that simulated approach shuffle only one word on each step, but here can be shuffled more than one word on each step.

- Word Ordering Network as syntactic word embeddings approach, which using LSTM (WON) [Nishida et al., 2017]. The WON receives a set of shuffled tokens and first transforms them independently to low-dimensional continuous vectors, which are then aggregated to produce a single summarization vector. They look at the word ordering task as a sequential prediction problem of a permutation matrix. They use a recurrent neural network (RNN) with long short-term memory (LSTM) units and a soft attention mechanism that constructs rows of permutation matrices sequentially conditioned on summarization vectors. Shortly, in this approach LSTM was used for permutation of words.

- Combinatory Categorial Grammar approach (CCG) [Zhang et al., 2012]. First assigning a lexical category to each word, and then recursively applying CCG rules bottom-up. They use tree with possible words connections and calculate some of the combinations with the best permutation score and also with the words as positive and negative examples for training model (online large-margin training). They used N-gram language model as decoder.

- Comparison of N-gram and LSTM model for permutation optimisation [Schmaltz et al., 2016]. The article compares different approaches for permutations and find out that LSTM works better with this task than N-gram language model [Zhang et al., 2012].

- Bag-to-sequence neural model approach inspired by attention-based sequence-to-sequence models [Hasler et al., 2017]. Takes seq2seq model and replacing the recurrent layer with non-recurrent transformations of the word embeddings. At each time step t they used beam search, which keeps n the best hypotheses according to scoring function S(·) using partial model score s(·) and estimates g(·).

- Pre-training BERT with permuted language model (PERT) [Cui et al., 2022]. PERT shares identical neural architecture with BERT, while there is a slight difference in the input and the training objective. The proposed PERT uses a shuffled sentence as the input, and the training objective is to predict the position of the original token.

- XLNet as a generalized autoregressive pretraining method [Yang et al., 2020]. XLNet maximizes the expected log likelihood of a sequence with using all possible permutations of the factorization order.

Now let’s look at some kaggle notebooks with solving for this task:

- Moving from the discrete set of permutations to the continuous set of bistochastic matrices and performing gradient descent in this space using reinforce [Relax, it’s Santa, kaggle, 2024]. They sample N permutations from, compute the perplexity of

each permutation, and update probability i,j in matrix in the direction that decreases the perplexity.

- Finding optimal or near-optimal solutions to complex problems by simulating the process of evolution [Genetic Search, 2024]. In each iteration of the algorithm, better-performing individuals (those with lower fitness scores) are selected to breed and create a new generation. The selection process often involves picking the top candidates (the best solutions) from the population.
- Using ChatGPT2 medium and lexical category to each word for finding minimum perplexity [Text Reordering LLM, 2024]. From this notebook we can find that it's possible to replace perplexity function from competition on perplexity from chat gpt, and calculations will be faster, but perplexity score will be completely different from our original function.

Most of the articles suggest beam search approach with some improvements or use all possible permutations on small sequences. Also, a lot of articles used LSTM model as the best model for choosing order of sequence. A lot of notebooks just use beam search with different improvements and optimisations.

Model Description

I take IBIS approach [Malkin et al., 2021] and change it, that it suited for this task. The main changes that I've done:

- replace gpt2 model from article on gemini-9B model, from our task and all connected details with it
- replace score function on our function, which count perplexity
- add possibility to work with batches for our score function
- change ibis code, that can be suited for our task: add start from current sequence, not from random, as option; rebuild permutation generating process Let's describe how this is working

First, I loaded a transformer-based language model for causal language modeling tasks. Before it was used gpt2, but I replace this on gemini-9B (gemma) (as it necessary for count perplexity function for this task).
The model has a decoder-only architecture pre-trained on a diverse and extensive dataset, enabling robust performance across a wide range of tasks.
Model Features:

- Architecture: A decoder-only model tailored for text-to-text tasks, such as reordering, generation, and summarization.

- Versatility: The model can handle token-level manipulations and complex logical reordering tasks, making it well-suited for applications requiring high linguistic precision.
- Optimization: The model supports half-precision (FP16) computations, enabling efficient use of GPU resources and faster inference.
- Pre-training: Trained on a vast dataset of text, the model can understand nuanced linguistic structures, ensuring that reordered sequences maintain semantic integrity.

The model was trained on different data sources, including web pages, code, mathematical and technical data. Extensive filtering processes ensure the exclusion of low-quality or harmful data, maintaining the reliability and ethical use of the model. Sensitive and private information is systematically removed to align with safety standards.

1. Environment Setup
   - loading the transformer model and its associated tokenizer
   - choose hyperparameters:
     - batch_size (b) - how many sequences we want to random generate for scoring initially (was using 16, 32, 64, 128 depending on size of sequence)
     - top_k (B) - number of candidate after shuffling, which we will be scoring (512)
     - max_steps - how many steps we plan to algorithm will be working (1024)
     - patience - how many steps we wait to early stop, if we don't have improvements (128)
   - choose loss function for score perplexity (cross entropy by the task)
2. Token Shuffling: A Probabilistic Approach (main shuffle function). The shuffle function orchestrates the reordering of tokens in a given sequence. It works by tokenizing the input text, applying constraints through a mask, and invoking an iterative optimization routine to generate reordered candidates.
   - The input sequence is tokenized, and special boundary tokens (e.g., BOS, EOS) are added to frame the sequence.
   - A mask identifies tokens that cannot be moved, ensuring semantic or syntactic constraints are respected.
   - The ibis function refines the sequence through an iterative search for optimal orderings based on token probabilities.
   - The function outputs step, when was improvements, type of improvement (init - initial value, GS - global search, LS - local search), perplexity for this step and the sequence
3. Scoring and Evaluation (score). Initially scoring function was different, but I rewrote it for current task, using current function of perplexity and also add compatibility with batches of sequences.

- Feeding the tokenized inputs through the model. This step generates the raw outputs (logits), representing the model's unnormalized confidence scores for each token in the vocabulary at each position in the sequence.
- The logits are transformed into log-probabilities using a log-softmax operation. This step normalizes the scores, making them interpretable as the model's predicted likelihood for each token. Negative log probabilities are computed to quantify the uncertainty of predictions.
- To evaluate the model's accuracy in predicting tokens, the predicted probabilities are shifted by one position. This alignment ensures that each token's prediction corresponds to its correct label in the sequence.
- A cross-entropy loss is calculated between the model's predicted probabilities and the ground-truth labels. This loss quantifies how well the model predicts the tokens at each position. The function averages the loss values over each sequence in the batch, resulting in a sequence-level loss score.
- The sequence-level loss is exponentiated to calculate perplexity. Perplexity indicates how "surprised" the model is by the sequence, with lower values signifying better performance.
- The function extracts token-specific scores by indexing the log-probabilities at the positions corresponding to the actual tokens in the input. These scores provide detailed insights into how confidently the model predicts each token.
- The token-level scores are summed for each sequence to produce an overall sequence score. This aggregated score reflects the model's performance over the entire sequence, enabling comparisons across batches or datasets.

Outputs:

- Sequence Perplexity: A measure of overall sequence predictability.
- Token-Level Scores: Log probabilities for each token in the sequence.
- Aggregated Sequence Scores: A single score representing the model's performance on the entire sequence.

4. Iterative Optimization Framework (ibis). The iterative nature of the ibis function means that the token sequence is refined step by step, with each iteration aiming to reduce the sequence loss (e.g., negative log-likelihood). By progressively improving the sequence, the function ensures that the final output is the optimal ordering or as close to optimal as possible.
   - Candidate Generation: At each iteration, the function generates a set of candidate reorderings by rearranging the tokens in the sequence. The candidates are created using two primary strategies:
     - Global Search: This involves large-scale rearrangements that explore the overall structure of the sequence. A set of candidate sequences is generated by applying diverse large-scale transformations to the current

sequence. Each candidate is scored based on a loss function (e.g., cross-entropy loss or perplexity). The candidate with the best score is selected for further refinement or iteration.

    - Local Search: This focuses on smaller, localized adjustments, refining token order within narrow windows. Operates within small, sliding windows (e.g., 3–5 tokens) of the sequence. A fixed-size window slides across the sequence, generating localized candidates by permuting tokens within the window. Each candidate is evaluated based on its loss score. The best candidate is chosen for further refinement or iteration.

- Evaluation: Each candidate reordering is evaluated using a scoring function, which calculates its loss (e.g., cross-entropy loss, perplexity). The scores indicate how well each candidate aligns with the model's learned probabilities and the constraints imposed by the mask.

- Selection: The most promising candidate—the one with the lowest perplexity selected as the updated sequence for the next iteration. If no improvement is observed for a specified number of iterations (defined by the patience parameter), the optimization loop terminates.

- Mask Update: The mask may be adjusted based on the fixed tokens and constraints to ensure consistency across iterations.

- Shuffle proposals

    - Candidate tokens identified in the search step are passed to the shuffle_proposals function, which generates and evaluates possible reshuffling proposals:
        - Each proposal represents a different arrangement of the candidate tokens.
        - Valid proposals (with sufficiently high scores) are applied to generate new token sequences.

- Update and Yield

    - Tokens are rearranged, and the resulting sequences are added to the batch for evaluation in the next iteration.
    - The function yields the best sequence found so far, its score, and the associated mask, enabling real-time monitoring of the optimization process.

Shuffling algorithms works like this (shuffle_proposals):

- The function takes a score matrix that represents pairwise relationships between tokens in a sequence and evaluates a series of reshuffling proposals. Its primary goal is to find the most optimal rearrangements of tokens while

adhering to constraints, such as maintaining strictly increasing order for specific combinations.

- The function begins by determining the sequence length, denoted as L. This length determines the possible reshuffling permutations.
- A tensor is created to represent permutations of indices up to the kopt dimension. This allows systematic exploration of possible reshuffling combinations for tokens.
- A □oolean mask ensures that selected indices satisfy constraints, such as being in strictly increasing order. This step avoids invalid or duplicate combinations by enforcing logical token sequencing.
- Current Order Score: Based on the token arrangement in the input sequence.
- Proposed Order Score: Reflecting the potential improvement when rearranging tokens according to a selected permutation.
- The score difference between proposed and current orders is computed, forming the basis for evaluating the quality of reshuffling proposals. Invalid combinations are penalized heavily to exclude them from consideration.
- The function identifies the top topk proposals with the highest score improvements. From these, bs proposals are randomly sampled for batching, ensuring diversity in the selected candidates.

The function returns:

- Indices: A batch of reshuffled indices, representing token arrangements for each proposal.
- Scores: The improvement scores for the selected proposals.
- Order: The specific reshuffling order used for each proposal.

The final sequence is still can be in the local optimum, not in global, but probability to be there is lower than for other approaches. Also, it can be launched some number of attempts and the global optimum eventually can be found.

Dataset

I take data from kaggle competition. For get data you need to use this command: [Kaggle santa, 2024]

```
kaggle competitions download -c santa-2024
```

We have 5 rows, each with words from Christmas stories, that we need to rearrange in correct order.

Row N	Length
0	10
1	20
2	20
3	30
4	50
5	100

Stories from 10 to 100 words. All words 225, unique words 94.

Text 1 includes Text 0. Text 3 includes Text 2. Text 5 includes all texts and is the most comprehensive. However, this describes the inclusion of text sets. The actual vocabulary in each text might differ, as a text can include another while still having unique words of its own.

If we create clusters for embeddings that we can see different clusters with prepositions (left), Christmas theme words (right), verbs (up). Seems that we can try to take words from each cluster, not repeated. More details in EDA notebook.

There is no test split in this dataset, because we can permutate text in any order to minimise perplexity, which can not cause an overfit.

Experiments

Metrics

Perplexity of text sequences - how well a model predicts a given sequence of text. It measures how uncertain a model is about predicting the next word in a sequence. Lower perplexity indicates better predictions.
As input it has permuted text and pretrained language model.
In this competition the model gemma-2-9b from google was used [https://www.kaggle.com/models/google/gemma-2/Transformers/gemma-2-9b/2].
The main process works this way:

- Adds special tokens (‘bos’ for “beginning of sentence” and ‘eos’ for “end of sentence”) to explicitly mark sequence boundaries.
- Tokenizes the text with the pre-trained tokenizer
- Put tokens in model and get logits
- Take logits as predictions for tokens in the sequence except the last one and take initial input tokens in the sequence except the first one as target. (shift that first token of prediction map to second token of input). It’s done because we have next-token prediction task, and we need to align correctly predictions with real values.

- Calculate Cross-Entropy Loss between shifted logits and input

Example:
Input: “bos This some sentence as example eos”
Logits (perfect situation) - predict next word from input word: “This some sentence as example eos”
Labels: “This some sentence as example eos”
Loss(Logits, Labels) -> min * Get perplexity as model’s uncertainty in predicting the sequence:

$$Perplexity = e^{loss}$$

So, finally we can say that perplexity quantifies how “confused” a language model is when predicting the next word in a sequence. Average of perplexities of each row is the final score.

Experiment Setup

There are some decisions for this task. We need to get permutations of each row somehow, check perplexity and if it’s lower than before (for example lower than 2178 on the raw texts), that it’s good iteration. I will try to find the best way to choose correct permutations of words.
Baselines:

- Brute force
- Greedy Decoding
- Beam Search
- Simulated Annealing

Approaches from articles, that I tried:

- IBIS approach [Malkin et al., 2021]
- Using LSTM for prediction next world in sequence [Nishida et al., 2017], [Schmaltz et al., 2016]
- Replace gemini model on PERT [Cui et al., 2022]
- Replace gemini model on gpt2 [Text Reordering LLM]

In all approaches I used this plan:

- Try approach on my data as is and measure perplexity for final sequence
- If it’s not so more than raw score (2178), than optimise this approach for my task
- Take best sequence from baseline (343.88) and try to improve with choosing approach

- Repeat search of better perplexity some amount of steps
- Add Simulated Annealing as up level

Because IBIS approach worked better, compare with others, that I describe hyperparameters for this approach:

- batch_size (b) - how many sequences we want to random generate for scoring initially (1st: 128, 2nd,3rd: 64, 4th:32, 5th: 16)
- top_k (B) – number of candidate after shuffling, which we will be scoring (512)
- max_steps - how many steps we plan to algorithm will be working (1024)
- patience - how many steps we wait to early stop, if we don't have improvements (128)
- repeat this approach with combination of simulated annealing 3 times

Baselines

- Brute force - just take all possible combinations of words and score each to find the lowest perplexity. The problem that this approach is too long because even in the first row where we have only 10 words there are 10! = 3628800 possible permutations for this and you need to score each. And the last row has 100! permutations. So, we need to find less time and resource consuming approach.
- Greedy Decoding - on each step choose the word that brings the lowest perplexity from words that we have not used before. It can work long by time because evaluate all candidates on each step, and also it can be local optimum decision, because you choose the optimal word on each step, but the combination of words from previous steps can not to be the most optimal choice.
- Beam Search - same as greedy, but instead of 1 best candidate it chooses k the best candidates on each step. It helps to do lower chances to be in the local optimove, because it's track k strategies on each step. But it is more computationally expensive than greedy decoding, as it evaluates and tracks multiple candidates at each step. Also, the results still can bring to the local optimum, instead global, if k was not enough.
- Simulated Annealing - start with random solution and make small changes from that to find better combination. We take two random words, swap it and hope to get better score. Repeat this some amount of steps. It's faster than brute force and trying to escape local optimum, but still cannot be global optimum. This method also sensitive to hyperparameters.

Row N	Raw	Random	Brute Force	Greedy	Beam Search	Simulated
0	3887.90	2105.57	1375.48	1327.97	515.99	496.23
1	6068.93	2833.36	-	2017.02	598.56	577.89
2	1118.26	-	-	967.78	526.17	394.08
3	1287.11	-	-	762.59	386.46	309.32
4	353.25	-	-	283.85	204.45	202.86
5	354.64	-	-	128.94	82.93	82.93
Mean	2178.35	-	-	914.70	386.02	343.88

- baselines/1_random (Raw / Random): First as baseline we can take perplexity of random texts as is (Raw). I tried to do random rearrange for row0 and row1, because they have the highest score, and this was improvements. We can use it as simple baseline and if other results will be higher, that something is definitely wrong.
- baselines/2_brute_force (Brute Force): After that let's try brute force for 0 raw (because it's smallest and has less possible combinations). But because it's time-consuming (n! permutations), I limited time by 2 hours, and tried to find the best combination for this time.
- baselines/3_greedy (Greedy): Next approach is greedy decoding. It's funny that for 6 seconds it's beat perplexity result for row 0 that was found for 2 hours :) With this approach we already get good perplexity, but it's only local optimum (optimum on each step independent of previous and next steps) and there is more fields for improvements. Good thing that it's working too much faster than brute force.
- baselines/4_beam_search (Beam Search): Beam Search was working 5 minutes on the row 0, but improving results in 2 times. For row 5 I stopped beam search after 1st attempt, because it was 2 hours only on one attempt. Beam Search showed huge improvement for all rows, that helps find more optimal but still local in some cases optimum.
- baselines/5_simulated_annealing (Simulated): After that I tried Simulated Annealing approach for hope to improve beam search results. This strategy with swap two random words can improve a little previous results.

The best mean perplexity from baselines is 343.88, that we can take as a baseline for experiments.

Results

First I take approaches from articles as is, and trying to launch this on my data. Here mean perplexity scores, that I get:

IBIS	LSTM	PERT	GPT2
452.98	3315.90	4569.11	3001.30

After that I found out that the most promising approach to try is IBIS, because all other approaches was too connected with model that you choose. I understood that use gemini model in this task is crucial for lower perplexity function, and you need to use all approaches with this tokenizer and model.
So I take IBIS approach and modify it for my task, after that added simulated annealing approach, and repeated it for 3 iterations:

Row N	IBIS (1)	IBIS + SA (2)	IBIS + SA (3)
0	496.23	496.23	496.23
1	549.27	536.55	511.98
2	308.11	308.11	303.34
3	266.65	236.24	242.79
4	126.45	126.45	121.13
5	50.89	49.62	49.62
Mean	299.60	292.20	282.97

The best score that I can achieve for 3 iterations is 282.97, which is much better, than baseline 343.88. If try more iterations that it can be improved even more, because I get global optimum only for 0 row. For other rows it's still local optimums based on kaggle leaderboard results.

Conclusion

I get task to rearrange sequence to get the lowest perplexity. This task mostly used in translation, grammar corrections.
I take dataset from kaggle, and evaluate it on the common baselines for this task.
After that I analysed different approaches from articles and also kaggle notebooks and tried them in this dataset.
Only one approach (IBIS) showed the promising results.
I take this approach as based and rewrite it and improved for current task.
After that I add simulated annealing approach from baseline, but it's not help to improve score.
Finally, I get good perplexity results, compare with baselines, using only IBIS approach with 3rd attempt.
It's possible to get better results, just increasing the number of attempts.
The IBIS approach works better and faster than beam search with combination of simulated annealing, but it still can stay in the local optimums.

References

1. [Kaggle santa, 2024] Ryan Holbrook, Walter Reade, Maggie Demkin, and Elizabeth Park. Santa 2024 - The Perplexity Permutation Puzzle. https://kaggle.com/competitions/santa-2024, 2024. Kaggle.

2. [Malkin et al., 2021] Nikolay Malkin, Sameera Lanka, Pranav Goel, Nebojsa Jojic. (September 2021). Studying word order through iterative shuffling 2021. Empirical Methods in Natural Language Processing, https://arxiv.org/abs/2109.04867

3. [Nishida et al., 2017] Noriki Nishida and Hideki Nakayama. (2017). Word Ordering as Unsupervised Learning Towards Syntactically Plausible Word Representations. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 70–79, Taipei, Taiwan. Asian Federation of Natural Language Processing. https://aclanthology.org/I17-1008/

4. [Zhang et al., 2012] Yue Zhang, Graeme Blackwood, and Stephen Clark. (2012). Syntax-Based Word Ordering Incorporating a Large-Scale Language Model. In Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 736–746, Avignon, France. Association for Computational Linguistics. https://aclanthology.org/E12-1075/

5. [Schmaltz et al., 2016] Allen Schmaltz, Alexander M. Rush, and Stuart Shieber. (2016). Word Ordering Without Syntax. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 2319–2324, Austin, Texas. Association for Computational Linguistics. https://aclanthology.org/D16-1255.pdf

6. [Hasler et al., 2017] Eva Hasler, Felix Stahlberg, Marcus Tomalin, Adrià de Gispert, and Bill Byrne. (2017). A Comparison of Neural Models for Word Ordering. In Proceedings of the 10th International Conference on Natural Language Generation, pages 208–212, Santiago de Compostela, Spain. Association for Computational Linguistics. https://aclanthology.org/W17-3531.pdf

7. [Cui et al., 2022] Yiming Cui, Ziqing Yang, Ting Liu, PERT: Pre-training BERT with Permuted Language Model. (March 2022). https://arxiv.org/abs/2203.06906?utm_source=chatgpt.com

8. [Yang et al., 2020] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le (january 2020). XLNet: Generalized Autoregressive Pretraining for Language Understanding, Carnegie Mellon University, Google AI Brain Team, https://arxiv.org/pdf/1906.08237

9. [Relax, it's Santa, kaggle, 2024] https://www.kaggle.com/code/simjeg/relax-it-s-santa

10. [Genetic Search, 2024] Basic Genetic Search Algorithm, https://www.kaggle.com/code/aatiffraz/basic-genetic-search-algorithm

11. [Text Reordering LLM, 2024] Text Reordering & Perplexity Calculation LLM Model, https://www.kaggle.com/code/shahzaibmalik44/text-reordering-perplexity-calculation-llm-model