Replacement Language Modeling

Bill Ray

November 17, 2022

1 Setup

1.1 Empty Token

A token sequence can be decoded into a string by concatenating the characters of the tokens. The empty token ([EMT]) is a special token that represents the empty string. Inserting the empty token into a token sequence at any point does not change the string that is decoded from the sequence. Token sequences that decode to the same string are functionally equivalent. Consider the empty token to be part of the vocabulary.

1.2 Normal Form and Replacement Process

Start with a sequence of tokens where every non-empty token is neighbored by empty tokens on both sides. Let's call this format the "normal form" of a sequence. Let the empty token have ID 0 in the vocabulary. Let all other tokens IDs correspond to non-empty tokens.

Example Sequence of token IDs in normal form: $[0, 23, 0, 14, 0, 18, 0]$

To perform the replacement process, replace zero or more tokens in the sequences with other tokens from the vocabulary. Notice that replacing a non-empty token with the empty token effectively deletes that token and replacing an empty token with a non-empty token inserts that token into the string that is decoded from the sequence. While replacement can change the length of the string that is decoded from the token sequence, the length of the token sequence is not changed during replacement.

Example Replacement: $[0, 23, 0, 14, 0, 18, 0] \rightarrow [0, 85, 54, 14, 0, 0, 0]$

After replacement, restore the sequence to normal form by replacing any consecutive runs of empty tokens with a single empty token and insert an empty token between any two non-empty tokens.

Example Normalization: $[0, 85, 54, 14, 0, 0, 0] \rightarrow [0, 85, 0, 54, 0, 14, 0]$

1.3 Insertions, deletions, and substitutions as replacements

Given a sequence in normal form:
$[0, 23, 0, 14, 0, 18, 0]$

An insertion can be made by replacing an empty token with a non-empty token:
$[0, 23, 0, 14, 0, 18, 0] \rightarrow [0, 23, 54, 14, 0, 18, 0] \rightarrow [0, 23, 0, 54, 0, 14, 0, 18, 0]$

A deletion can be made by replacing a non-empty token with an empty token:
$[0, 23, 0, 14, 0, 18, 0] \rightarrow [0, 23, 0, 0, 0, 18, 0] \rightarrow [0, 23, 0, 18, 0]$

A substitution can be made by replacing a non-empty token with another non-empty token:
$[0, 23, 0, 14, 0, 18, 0] \rightarrow [0, 85, 0, 14, 0, 18, 0]$

1.4 Levenshtein edit distance

The Levenshtein distance is defined between two sequences as the minimum number of edits (insertions, deletions, and substitutions) required to change one sequence into another.

For example, the character-level Levenshtein distance between "kitten" and "sitting" is 3, since the following 3 edits change one into the other, and there is no way to do it with fewer than 3 edits:

1. kitten → sitten (substitution of "s" for "k")
2. sitten → sittin (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end)

1.5 Optimal Replacement Policy

Given two sequences of tokens, an initial sequence, and a target sequences the optimal replacement policy can identify all edits (insertions, deletions, and substitutions) that, when applied to the initial sequence, reduce the Levenshtein distance between the initial sequence and the target sequence.

A few-hot vector is a binary vector dominated by zeros but may contain one or more '1' values. Few-hot vectors represent classification labels where multiple classes are equally valid.

When given an initial sequence and target sequence in normal form the optimal replacement policy returns a few-hot vector for every token in the initial sequence. These few-hot vectors have the same length as the vocabulary. Together these vectors make up a $|x|$ by $|v|$ matrix where x is the the initial sequence and v is the vocabulary. This few-hot matrix R has elements $R_{i,j} = 1$ if setting the token at position i in the initial sequence to the token with token ID j causes an edit that reduces the Levenshtein distance between the initial sequence and the target sequence ignoring empty tokens. $R_{i,j} = 1$ also if j is the token ID at position i and there are no possible replacement tokens for position i that would reduce the Levenshtein distance. Otherwise $R_{i,j} = 0$.

2 Replacement Language Modeling

Language modeling tasks involve predicting a probability distribution over the vocabulary for every token in the input sequence. In masked language modeling the loss is calculated between the predicted probabilities at the masked positions and a one-hot vector encoding the true token at the masked position. In causal language modeling the loss is calculated between the predicted probabilities at the last position and a one-hot vector encoding the true token at the next position.

I propose replacement language modeling which calculates the loss between the predicted probabilities at every position and a few-hot vector produced by the optimal replacement policy. To accomplish this I suggest starting with a token sequence from a text dataset and performing edits on the sequence to corrupt it. Apply the optimal replacement policy to the corrupted sequence and the original sequence to produce a few-hot vector for every token in the sequence. These vectors encode all possible edits that reduce the Levenshtein distance between the corrupted sequence and the original sequence. Next, feed the corrupted sequence into a language model and calculate the loss between the predicted probabilities at every position and the few-hot vectors produced by the optimal replacement policy.

Example

1. Begin with a sequence from the training data: "I cooked dinner for my family"
2. Corrupt the sequence by applying insertions, deletions, and substitutions: "I cooked dinner for my family" $\rightarrow$ "I dinner for" $\rightarrow$ "I car dinner for"
3. Normalize the sequence by adding the empty token before and after every token: "[EMT] I [EMT] car [EMT] dinner [EMT] for [EMT]"
4. Use the optimal replacement policy to identify what replacements reduce the Levenshtein distance to the original sequence ignoring empty tokens
    - [EMT] = { [EMT] = 1, 0 elsewhere }
    - I = { I = 1, 0 elsewhere }
    - [EMT] = { [EMT] = 1, 0 elsewhere }
    - car = { cooked = 1, 0 elsewhere }
    - [EMT] = { [EMT] 1, 0 elsewhere }
    - dinner = { dinner = 1, 0 elsewhere }
    - [EMT] = { [EMT] = 1, 0 elsewhere }
    - for = { for = 1, 0 elsewhere }
    - [EMT] = { my = 1, family = 1, 0 elsewhere }

5. Use a language model to predict a probability distribution over the vocabulary at every position in the sequence

6. Update the language model to optimize a loss between the predicted probabilities and the few-hot vector at every position in the sequence. The loss may be more interpretable if label vectors at every timesteps are normaliezd to sum to 1.

This training method can be viewed as behavior cloning of the optimal replacement policy, except the language model is not given the target sequence.

2.1 Increased sequence length

One drawback of replacement language modeling is the requirement that sequences are in normal form so that any edit can be expressed at any position. Over half the tokens in a sequence in normal form are empty tokens, meaning the context window size of the model is effectively halved. This problem may be mitigated by dropping some of the empty tokens from the sequence; however, this would reduce the editing options that can be expressed as token replacements.

2.2 Calculating Perplexity

The perplexity of a sequence given to the model can be calculated from the models output matrix Y:

$$\text{PPL(Y)} = \exp\left(-\frac{1}{|x|}\sum_{i=1}^{|x|}\log Y_{i,x_i}\right) \tag{1}$$

Where x is the sequence of token IDs, v is the vocabulary, and Y is an $|x|$ by $|v|$ matrix where $Y_{i,j}$ is the probability of token j at position i. The perplexity is a measurement of how likely the language model considers a sequence of tokens.

3 Guided Corruption

I propose guided corruption, a method of corrupting text samples from the training data in a way that may provide more effective training samples to the model. Guided corruption generates training samples throughout the training process based on the current state of the model. Guided corruption applies edits to sequences in the following way:

1. Begin with a sequence from the training data

2. Apply the optimal replacement policy to the sequence to generate a few-hot matrix R

3. Apply the language model to the sequence with the current weights to generate a prediction matrix Y

4. Generate a new probability distribution vector P_i over the vocabulary for every token in the sequence such that $P_{i,j} = 0$ if $R_{i,j} = 1$ and $P_{i,j} = Y_{i,j}$ otherwise

5. Renormalize P so all rows sum to 1

6. Randomly select a position i in the sequence and replace the token at position i with a token sampled from the probability distribution in row i of P

7. Repeat this process but begin with the sequence generated in the previous step

Guided corruption selects the corruption edits to be edits that the language model predicts given that the edits increase the Levenshtein distance from the original sequence. I hypothesize guided corruption would generate corrupted samples with perplexity lower than that of randomly corrupted samples. This would make the difference between the corrupted sample and the original sample be the difference between what the model incorrectly thinks has high probability and what sequence actually is in the training data. Perhaps training on these sequence pairs would focus on "misconceptions" the model has and more efficiently train the model.

One drawback of guided corruption is it requires the model to process the sequence one time for every corruption step which is far more computationally intensive than randomly corrupting the sequence. Perhaps a combination of random corruption and guided corruption would be effective and more computationally efficient.

4 Text Generation

A replacement language model can be used to generate unconditioned text in the following way:

1. Begin with a sequence containing only the empty token

2. Apply the language model to the sequence to generate a prediction matrix Y

3. Randomly select a position i in the sequence and replace the token at position i with a token sampled from the probability distribution in row i of Y

4. Repeat this process but begin with the sequences generated in the previous step until the generated sequence has sufficiently low perplexity according to the model

Conditional text generation can be accomplished by converting an input sequence into normal form and then following the unconditional text generation procedure starting with the normal form sequence. Optionally, if editing the original sequence is undesired, modify step 3 to never sample a position inside the original sequence.

It may be possible to reduce the number of empty tokens in the sequence by removing empty tokens from the sequence after the replacement process if those empty tokens were not replaced. This would reduce the number of tokens in the sequence and increase the number of non-empty tokens that can fit in the context window of the model, potentially enabling the model to generate longer strings. However, this would reduce the number of editing options that can be expressed as token replacements.

5 Confidence Position Selection

I propose confidence position selection, a method of selecting the position to edit in a sequence based on the confidence of the model. Instead of randomly sampling a position in the sequence i, select i to be

$$i = \arg\max_i(\max_j Y_{i,j}) \tag{2}$$

where Y_i is a probability distribution over the vocabulary at position i in the sequence.

Confidence position selection can replace the random position selection in text generation and guided corruption. Confidence position selection may generate higher quality text because the edits to the sequence are sampled from a probability distribution with lower variance and tokens the language model assigns high probability to are more likely to be sampled. This may result in text with lower perplexity which could improve the text generation capability of the model and further lower the perplexity of the training samples generated by guided corruption.

6 Benefits of Replacement Language Modeling

Replacement language models may be able to edit text more effectively than causal or masked language models in tasks such as post-editing, grammar or spelling checking, and perhaps code debugging.

Observing the behavior of a replacement language model may provide insight into how language models understand text by observing how the language model replaces tokens in the sequence and, in the case of confidence position selection, in what order.

Because replacement language models can generate text conditioned on a prefix and a suffix, they may give human prompt engineers more control over the text generation process. For example a prompt engineer could define how the sequence starts, a few words that must be contained in the middle and how

the sequence ends. A replacement language model may be able to effectively generate text around these constraints.

This ability to fill in gaps may allow a replacement language model to exhibit planning behavior previously unseen in language models. A replacement language model would theoretically have the capability to produce an outline of what it was going to generate and then fill in tokens between the outline or even write over the outline altogether. It is unlikely a replacement language model would learn any planning behavior from simply the replacement language modeling task because this task only trains it to identify immediate actions to reduce the Levenshtein distance; however, if a replacement language model is trained on an autoregressive text generation task, where its outputs become its inputs, it may learn to plan.

7 Reinforcement Replacement Language Modeling

Likely replacement language modeling alone would not be enough to endow a language modeling with planning behavior. I hypothesize a reinforcement learning objective would be required to train a language model to plan. I propose the following reinforcement learning environment:

- **Initial State Distribution:** Sample a sequence of tokens from the training data
- **States:** A sequence of tokens in normal form
- **Actions:** A prediction matrix Y over the sequence
- **Transition Function:** Sample from the prediction matrix and apply an edit to the sequence through the replacement process
- **Episode Termination:** The sequence has sufficiently low perplexity according to the model
- **Rewards:** Rewards applied when the episode terminates. Rewards could be any evaluation metric of the sequence, for example:
  - Negative Levenshtein distance to a target sequence
  - Neural similarity to a target sequence
  - Human evaluation of the sequence
- **Discount Factor:** If rewards are always positive a discount factor less than 1 would incentivize the model to generate shorter text. If rewards are negative a discount factor of 1 would incentivize the model to generate longer text.

I hypothesize a replacement language model could be effectively fine-tuned with a reinforcement learning objective on a downstream task. I also believe such fine-tuning may cause planning behavior unseen in language models pretrained with other language modeling tasks. Because the outputs of the model affect the inputs of the model at the next time step and rewards are assigned several time steps in the future, the agent is incentivized to produce outputs that cause the language model to produce future outputs that will receive high rewards. This incentive structure may manifest as chain of thought prompting or even outline generation, in other words, planning.

8 Questions

1. Have past works explored the idea of replacement language modeling?
2. Have past works trained classifiers with few-hot vectors, were there any tricks needed to make that work?
3. Do causal language models trained with a reinforcement learning object ever exhibit chain of thought prompting behavior?
4. What experiments could most effectively test these ideas?
5. Were there any ideas that seemed interesting or useful?
6. Were there any ideas that seemed like they were built upon misconceptions or misunderstandings?
7. What ideas do you have related to this topic?