Substrings, Subsequences, and Temperature

May 2, 2024

1 Introduction

First, I'll define substring and subsequence. Let $w = [w_1, w_2, \ldots, w_n]$ be a list of words. Let s also be a list of words. We say w is a *substring* of s if w occurs contiguously in s. We also say w is a *subsequence* of s if w occurs possibly non-contiguously in s (extra words can appear between w_i and w_{i+1} within s).

So as an example, if we have $w =$ "pizza is delicious".

- If $s =$ "I think pizza is delicious", then w is a substring of s (and also a subsequence).
- If $s =$ "The pizza we ordered is delicious", then w is a subsequence of s but not a substring.

On a hypothetical level, it seems that both substring and subsequence relationships are important to be able to understand in order to model natural language sentences. Furthermore, this kind of abstact relationship/pattern can also occur in other domains, too.

I find the substring and subsequence relationships interesting because they are very similar in spirit, but actually differ on a computational level. Generally, substring is simpler than subsequence – in subsequence we need to model longer range dependencies. However, while poring through the mathematics behind how transformers work, it seems on first glance that it is easier for me to set a transformers' weights to recognize subsequences than to make them recognize substrings. This feels slightly puzzling to me, as I would have expected substring to usually be easier to express.

Without going into excessive detail, the heart of the matter seems to lie in the attention layer where the attention weights at each position are normalized using softmax. If we wanted to recognize substrings, we want to be able to say there is a certain word in an exact position – and to exactly express this we need to be able to focus all our attention weight on an exact single position. The softmax distribution spreads attention weight around, so we may focus on one symbol here and another symbol preceding it – which is like how a subsequence works.

In short, I think the standard transformer is good at subsequence relationships – It can express them exactly. But the non-discreteness of softmax (as

opposed to argmax) suggests it can only approximate substring relationships, so the performance would degrade on larger input lengths.

I want to see if these speculations apply when actually training transformers.

2 Set Up

So there are three questions I wanted to ask

1. Is substring harder than subsequence for transformers?
2. Lowering the softmax temperature makes the distribution more discrete (we can derive that a temperature of $\frac{1}{n^3}$ makes it discrete enough to represent substrings). Can a transformer still learn well with this temperature?

To answer these questions, I'll try it transformers on both a synthetic and a natural task.

2.1 Formal Language Recognition

I'll train a transformers to recognize if a string contains 'abc' as a substring. Using formal language notation, we say that $(\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*)$ is the subsequence task and then $(\Sigma^* abc \Sigma^*)$ task. I'll also train a transformer using the same hyperparameters to see if a string contains 'abc' as a subsequence, and compare their accuracies.

Then, I'll try the same hyperparameters, but using temperature of $\tau = \frac{1}{n^3}$, which lowers the temperature more and more the longer the input is. This will see the effect of temperature on the substring/subsequence relation.

2.2 Movie Review Sentiment Analysis

The synthetic data might be too contrived. I also will check the effect of the $\tau = \frac{1}{n^3}$ temperature on a more natural task. In theory, substrings and subsequences are also relevant for this task. For instance, the phrases "not good at all" and "not bad, but really good!" both contain "not good", but one is a substring and one is a subsequence. The substring one is negative, but the subsequence one is actually positive! So it seems important for a model to be able to represent both substrings and subsequences.

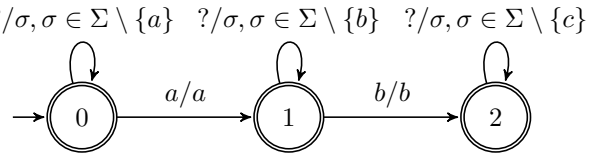
3 Some Preliminary Experiments

I first tried out the subsequence $(\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*)$ language and the substring $(\Sigma^* abc \Sigma^*)$ language.

3.1 Data Generation

Since this is a synthetic task, I had to generate data. Here is what I did. I defined functions to sample strings from the given language, and its complement, within length bounds l_1 and l_2

- SUBSTRING_POS(l_1, l_2) uniformly samples a length $l \in [l_1, l_2]$, then uniformly samples $w \in \Sigma^l$ until it finds a $w \in \Sigma^* abc \Sigma^*$.
- SUBSTRING_NEG(l_1, l_2) uniformly samples a length $l \in [l_1, l_2]$, then uniformly samples $w \in \Sigma^l$ until it finds a $w \notin \Sigma^* abc \Sigma^*$.
- SUBSEQUENCE_POS(l_1, l_2) uniformly samples a length $l \in [l_1, l_2]$ (with the restriction that $l \geq 3$), then uniformly samples $w \in \Sigma^l$. Then, three positions $i < j < k$ are uniformly sampled, and then we set $w_i = a$, $w_j = b$, and $w_k = c$.
- SUBSEQUENCE_NEG(l_1, l_2) uniformly samples a length $l \in [l_1, l_2]$. Then, $?^l$ is fed into the following transducer



3.2 Training and Test Distributions

Using the above generating functions, I sampled some decently sized training and test sets. The alphabet I used was $\Sigma = \{a, b, c, d, e\}$.

Subsequence Sets and Bin Sizes

Set	0-50	50-100	100-150	150-200
subsequence-train-pos	10k	-	-	-
subsequence-train-neg	10k	-	-	-
subsequence-test-pos	10k	10k	10k	10k
subsequence-test-neg	10k	10k	10k	10k

The substring distribution is exactly the same, so I copy the table here.

Substring Sets and Bin Sizes

Set	0-50	50-100	100-150	150-200
substring-train-pos	10k	-	-	-
substring-train-neg	10k	-	-	-
substring-test-pos	10k	10k	10k	10k
substring-test-neg	10k	10k	10k	10k

3.3 Training Details

The task here is to read in a string, and either accept or reject it from the given language. This is achieved by appending an EOS token to the end of the word, running the word through the transformer, adding a sigmoid layer at the end, and accepting iff the output on EOS is > 0.5

For both the substring and subsequence tests, I have the training set as the full 20k training samples of up to length 50. Thus, half are postive and half are negative samples. I split the training set into 16k for training and 4k for validation. The loss function I used was binary cross-entropy.

The transformers used future masking, no positional encodings, and a learned word embedding. I used the following hyperparameters.

- 1 epoch
- lr = 0.0001
- 2 layers (SA+FFN alternation)
- 5 hidden dimensions

The test set is all 40k samples, across 4 bins of lengths not seen during training. It only took the transformer 1 epoch to basically get perfect performance on recognizing subsequence which is why I used that as the baseline to compare with substring. Finally, I also added a run where the temperature is manually set to $\frac{1}{n^3}$.

3.4 The First Run

Again, all hyperparameters and training setups are the same in the below three rows, except they use different training languages and different temperatures. I

Experiment	Validation	Bin-1	Bin-2	Bin-3	Bin-4	Test
$\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$	1.00	0.9998	1.000	1.000	1.000	0.995
$\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*, \tau = \frac{1}{n^3}$	0.999	1.00	1.004	1.00	1.00	0.995
$\Sigma^* abc \Sigma^*$	0.651	0.644	0.535	0.510	0.501	0.541
$\Sigma^* abc \Sigma^*$, $\tau = \frac{1}{n^3}$	0.499	0.489	0.506	0.501	0.499	0.494

4 Expanding Scope

I feel like the subsequence task is too easy, based on the distribution. All strings without the subsequence abc will have very few c's in them based on how I generated them. Thus instead of having to check for the subsequence, the simple heuristic of if there are c's at the start of the string but not the end is sufficient.

To make it less trivial, I ran some more tests with an expanded alphabet. Now I have Σ to be all 26 ASCII letters 97 to 122.

4.1 Setup Details

The training and test distributions are exactly the same as above. The transformer set up is also the same, but with the new hyperparameters as follows

- 50 epoch
- lr = 0.0001
- 4 layers (SA+FFN alternation)
- 10 hidden dimensions

It is slightly less trivial to perfectly learn subsequence, which is why we have 50 epochs as the baseline. I also added more dimensions and layers. I suspect that more layers makes it easier to express subsequence/substring things, since each attention layer allows us to naturally do a sort of ordering on positions.

4.2 More Results

Using the setup described above, I gave the subsequence task more opportunity to learn. However, it seems that performance still degrades when tested on strings outside the lengths encountered in the training set.

Language	Validation	Bin-1	Bin-2	Bin-3	Bin-4	Test
$\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$	1.00	1.00	1.00	1.00	1.00	0.995
$\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*, \tau = \frac{1}{n^3}$	1.00	1.00	1.00	1.00	1.00	0.995
$\Sigma^* abc \Sigma^*$	1.00	0.999	0.997	0.985	0.956	0.974
$\Sigma^* abc \Sigma^*, \tau = \frac{1}{n^3}$	0.922	0.925	0.683	0.547	0.508	0.655

5 IMDB Movie Reviews

This is a more natural task, but also in the same spirit as the synthetic one. It's a binary classification task, and presumably being able to model substrings/-subsequences is helpful for it.

5.1 Data

The data was taken from `https://ai.stanford.edu/~amaas/data/sentiment/`. There are 50k Movie Reviews, 25k of which are positive and 25k of which are negative. As suggested, these were split evenly into a training set of 25k with 12.5k positive and 12.5k negative, and the test set similar. There is no overlap between the splits.

5.2 A Small Baseline (part 3)

I used a transformer encoder, with dimension 256, with future masked attention (each position only looks at previous ones), with 4 alternations of self-attention and feed-forward layers, and cross-entropy loss using two classes (positive and negative). Each word in the vocabulary is represented using a learned word-embedding that maps words to vectors.

In training, I did 2.5k training set with 10 epochs and a learning rate of lr = 0.0002. The training set was furthermore split into 2000 for training and 500 for validation. I used the adam optimizer.

Train	Validation
1.00	0.782

There were some significant problems. First, I kept having CUDA memory errors, so even though 25k training samples were available, I only trained on 2.5k. Furthermore, the validation set was only 1k, as I kept having problems. The lack of data could explain why the model did not learn super well – it performs at random chance on the validation set. Though, it did seem to do better on the small validation set I reserved to check at the end of every epoch.

Anyways, it seems that the model is essentially memorizing the training set, as it achieves perfect accuracy, and performs at chance on the validation set. So the most immediate room for improvement is using more data.

While I was messing around with different parameters, I found that having more than 1 layer helped, but it was hard to tell whether adding more beyond that made a difference, as I'd have to tweak the learning rate again to make it perform well. Perhaps having more layers would lead to better performance in the future. Also, the dimension of 256 was completely arbitrary, and I didn't increase it in this baseline because it made it run slower.

5.3 Setup Details

After solving some of these problems, I played around with the parameters for a while and set up the final experiments.

The setup is identical to the previous ones, except I pre-processed the movie reviews by removing special characters and turning everything lowercase. The hyperparameters are as follows:

- 10 epoch
- lr = 0.000008
- 1 layers (SA+FFN alternation)
- 32 hidden dimensions

5.4 Results

I think it kept overfitting on the training data, so I kept reducing the size of the model. Nevertheless, it seemed to make no difference on the test set.

Experiment	Train	Validation	Test
Regular	0.968	0.854	0.5
$\tau = \frac{1}{n^3}$	0.652	0.645	0.5

6 Concluding Remarks

The experiments seem to affirm our intution that transformers may have a harder time with substrings as opposed to subsequences. It takes far fewer layers, dimensions, and training epochs for a transformer to learn subsequences as opposed to substrings. Furthermore, even given a more time and resources, transformers still have degraded performance on strings of longer lengths than seen in training, as suggested in section 4.2.

The potential modification of having a input-size dependent temperature, which in theory could help a transformer select positions in a discrete manner, doesn't actually help in practice. It seems to degrade accuracy significantly, as suggested in section 3.4. section 4.2 even when provided enough time to perform well on the training set, the results do not extend to higher string lengths. Thus having the temperature $\frac{1}{n^3}$ is not a viable modification.

This is furthermore affirmed by the runs on the movie sentiment classification task. Even though no conclusions can be made due to the inability of the trained models to generalize beyond the training set, it is at least clear that the low temperature severely impacted the training process of the model, seen by the degraded accuracy in section 5.3.

Perhaps future analysis into substrings vs subsequences, and where these phenomena occur "in the wild", could inform our understanding of how transformer models function and learn.

7 Running the Code

The models are all in the models folder. This can be found at this share link

Model	Description
baseline.pt	The baseline model
movie.pt	Trained on movie task
movie_temp.pt	Trained on movie task and $\tau = \frac{1}{n^3}$
subseq_small.pt	Trained on subsequence task, few epochs
subseq_small_temp.pt	Trained on subsequence task, few epochs and $\tau = \frac{1}{n^3}$
substr_small.pt	Trained on substring task, few epochs
substr_small_temp.pt	Trained on substring task, few epochs and $\tau = \frac{1}{n^3}$
subseq_large.pt	Trained on subsequence task, many epochs
subseq_large_temp.pt	Trained on subsequence task, many epochs and $\tau = \frac{1}{n^3}$
substr_large.pt	Trained on substring task, many epochs
substr_large_temp.pt	Trained on substring task, few epochs and $\tau = \frac{1}{n^3}$

To evaluate the abc models, do

```
> python3 eval_abc_substr.py substr_small.pt
> python3 eval_abc_substr.py substr_small_temp.pt
```

Similarly, to evaluate the movie models, do

```
> python3 eval.py movie.pt
> python3 eval.py movie_temp.pt
```