Università della Svizzera italiana

Institute of Computing CI

Deep Learning Lab **2021**

Student: Naga Venkata Sai Jitin Jami

Solution for Assignment 3

1. Language Modeling with RNNs

1.1. Text Data

Properties of data can be found in table 1

Character vocabulary size	107
Number of lines	5033
Total number of words	32589
Number of characters	138881
Number of upper case character	9666

Table 1: Text data properties

If asked, further pre-processing of the text would involve making all characters lowercase and reducing the number of unique characters by getting rid of all the special characters like "&", "%" but keeping important characters like punctuation marks..

1.2. Batch Construction

1.2.1. Q1

The `if` branch exists to check if the word is actually in the vocabulary or not. There is an `extend_vocab` flag that would add the word to the vocabulary if it does not exist in the vocabulary.

1.2.2. Q2

The dictionary `id_to_string` has the `id` as the key and `string` as the value. But in `string_to_id` has the `string` as the key and `id` as the value.

1.2.3. Q3

The method `__len__` returns the number of characters including spaces and breaks.

1.2.4. Q4

The method `__len__` returns the number of batches created by `create_batch`.

1.2.5. Q5

The first line: `padded = input_data.data.new_full((segment_len * bsz,), pad_id)` is creating a new tensor `padded` of the size `segment_len * bsz` filled with `pad_id`.
The second line: `padded[:text_len] = input_data.data` is filling up the values of `padded` with `input_data.data`. Since the size of `padded` is larger than the size of `input_data.data`, the remaining elements will be 0.

1.2.6. Q6

The size of `padded[i * bptt_len:(i + 1) * bptt_len]` is `[bptt_len - 1, bsz]`.

1.2.7. Q7

The size of `padded[i * bptt_len - 1:(i + 1) * bptt_len]` is `[bptt_len, bsz]`.

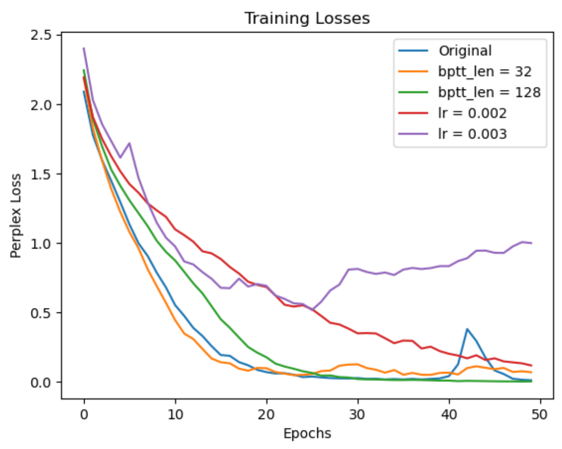
1.3. Model and Training

"Aesops Fables" from the Project gutenberg website was used as the training text. An LSTM model was modelled with the following parameters:

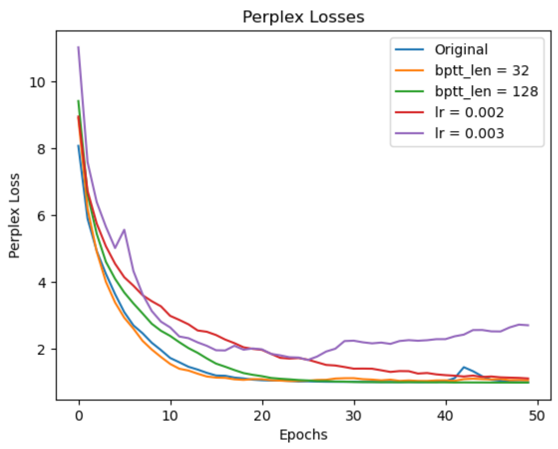
- Embedding Layer of dimension 64
- 1 LSTM layer of 2048 elements
- Back progation through time (BPTT) span of 64 characters
- Adam optimizer with learning rate = 0.001
- Gradient clipping with a clipping threshold of 1.0

The above model was run of **50 epochs**. Training and perplex loss was tracked to check performance after ever epoch. The following parameters were also changed to check for performance. The performance plots can be found on figure 1

- BPTT Span: 32, 128
- Learning Rate: 0.002, 0.003



(a) Training Losses



(b) Perplex Losses

Figure 1: Performance Plots

The main model has achieved `perplex_loss = 1.01` after 50 epochs. We can see in the performance plots that the original configuration and `bptt_len = 32` have identical evolution of losses,

training and perplex. `bptt_len = 128` eventually reaches similar accuracy as the ones before but takes longer to get there.
In the cases where learning rate was changed, there is a drastic change in performance. In the case of `lr = 0.003`, the optimization seems to be diverging and getting worse with every epoch after 25. Same can be observed in perplex loss plot in figure 1b.
Apart from tracking the losses, the model performance was tracked by using it to predict 5 characters after the prompt "Dogs like best to" at every epoch. Since only 5 characters were chosen, it was hard to track any meaningful increase in performance.

1.3.1. Testing

For the prediction function, 2 predict methods were implemented, greedy and sampling. When the output of the model is put through a softmax layer, we get the probabilities of all the characters in the vocabulary The original model was tested against the following prompts. The model was used to predict the next 1000 characters.

- Original prompt from the book: "The Farmer and the Stork"
- Prompt that is similar to the ones in the book: "The Elephant and the Wagoner"
- Text in similar style: "The Student and the pan"
- Random but sensible text: "A bread is never eaten"

Thoughts on the text generated:

- Original Prompt: The greedy prediction and sampling prediction generated text that is in English, however grammatically not always correct. Since the original prompt was a part of the training set I expected better, almost exact, output to the one in the text but this was not the case. However, greedy prediction had much better output compared to sampling prediction.
- Similar Prompt: As was the case before, both predictions were proper English but with inaccurate grammar. However, in this case sampling prediction performed better than greedy prediction. Greedy prediction essentially picked up a chunk from the text and then it reproduced just the text outright. I believe this is because once it picks up on the text it goes down the same path without any chance of choosing a less probable character as the next character.
- Similar style text: As was the case before, both predictions were proper English but with inaccurate grammar. Greedy prediction performed better but I can't help but imagine it has done the same as before. Where it picked up a similar prompt from the text, kept feeding the same text again and again to reproduce a prompt from the text.
- Random: Greedy prediction has the tendency to predict the same text every time it sees an unknown text prompt. Since the prediction is awfully similar to the one from the previous point.

My initial thoughts were that greedy predictions would perform the best. But after looking at the results its evident that greedy predictions have a tendency to fall into rabbit holes. It latches onto a random part of the existing text and starts reproducing as is from there on out. In my opinion, sampling prediction is better than greedy prediction.

2. Questions

2.1. Q1

Perplexity of prediction of uniform probability of all characters in a vocabulary size of `V` = `V`.

2.2. Q2

Applications of LSTM networks to solve sequential problems:

- Time-series forecasting
- Hand writing generation

2.3. Q3

Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. Similarly, vanishing gradient are a problem where the updates to neural network weights is extremely small. This leads to long term learning dependencies in the model because the optimization problem becomes either too sensitive or "stubborn" to change. The model is either too sensitive so it can't learn long term dependencies because it unlearns everything and learns everthing at every gradient step. Similar problem occurs with vanishing gradient where the model is not learning enough.