Homework3 Report

Yang Xiaocong

(To reproduce experiment results in this report, refer to README.md in codes file submitted.)

1. Basic Models

1.1. RNN

Our RNN model achieves a perplexity score of 134.69 in test set, by using following architecture and hyperparameters: LSTM cell, two layers of unidirectional rnn, 200 of input size and hidden size, 0.2 of dropout rate and 2 of learning rate. Training and validation curves are below:

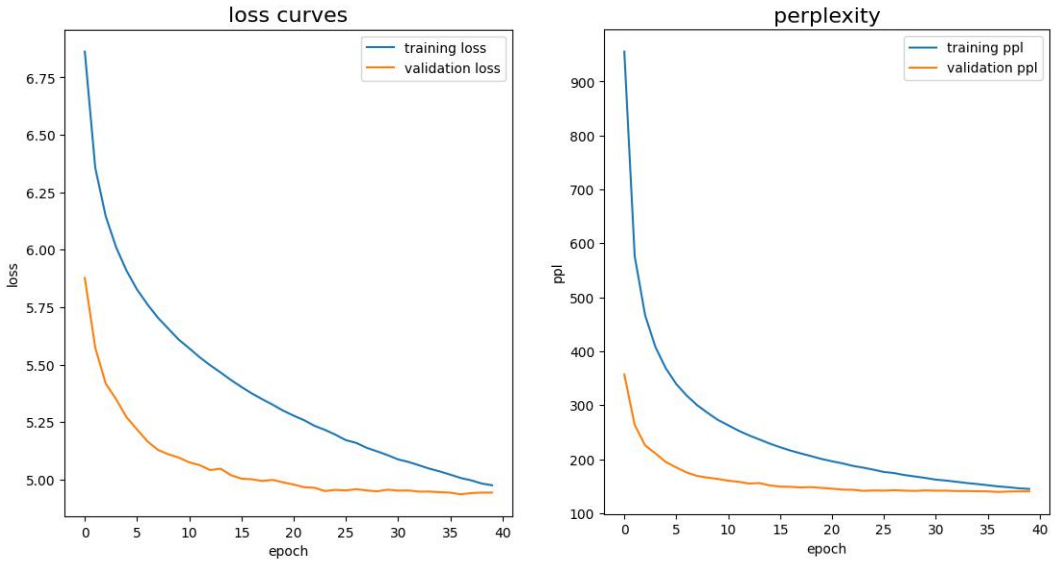

Figure 1 Loss and perplexity curves of RNN model

Complete training log can be retrieved in *'training_logs / screenlog_RNN.log'* file.

1.2. Transformer

Our Transformer model achieves a perplexity score of 199.58 in test set, by using following architecture and hyperparameters: two layers of standard transformer stack with four heads, 200 of input size and hidden size, 0.5 of dropout rate and 0.1 of learning rate. Training and validation curves are below:

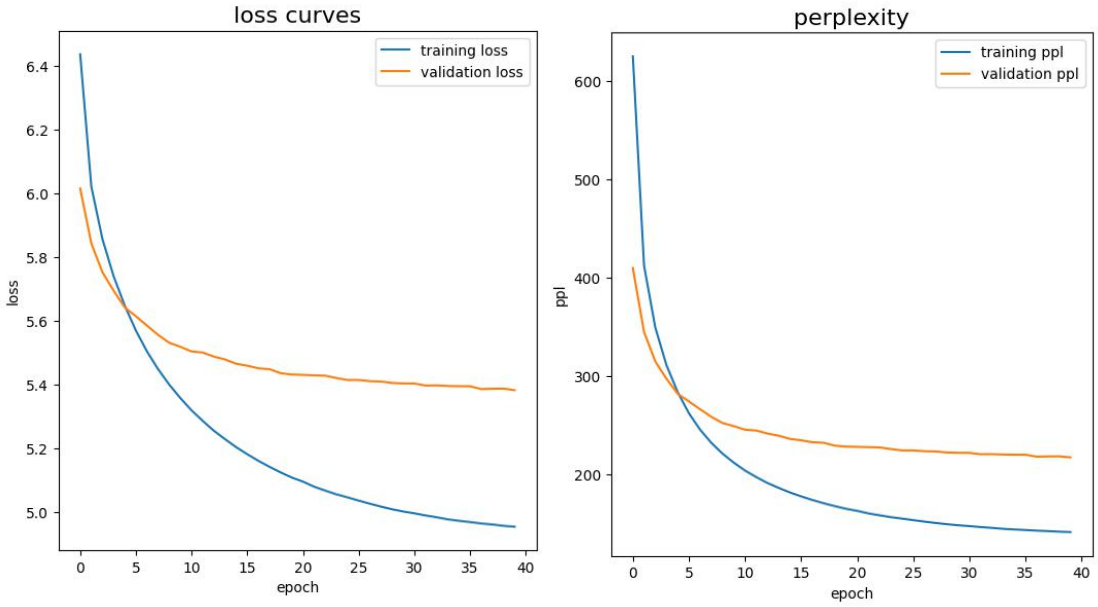

Figure 2 Loss and perplexity curves of Transformer model

Complete training log can be retrieved in *'training_logs / screenlog_Transformer.log*' file.

2. Data preparation

2.1. Text preprocessing for language modelling

In the context of transformer models, text preprocessing refers to the pipeline where raw input strings are sequentially normalized (e.g. stripping accents and whitespaces, lowercasing, etc), pre-tokenized (splitting a text into smaller objects that give an upper bound to final tokens), tokenized (training on the pre-tokens in corpus

to generate token vocabulary and mapping tokens to IDs, each ID corresponding to a randomly initialized embedding vector to be learned when training the language model) and post-tokenized (adding some special tokens such as [CLS] and [SEP], padding and truncating the sentences).

The code in *data.py* follows similar logic as standard text preprocessing pipeline in transformer models. Function *'get_tokenizer'* splits text into words and vocabulary is constructed through *'counter.update'* . Then tokens are mapped to IDs in function *'data_process'*, and finally function *'batchify'* split sentences (now represented by sequence of IDs) into 'bptt'-long subsentences to feed into language model.

2.2. Image preprocessing for object recognition

In traditional digital image filed, image preprocessing refers to the process removing irrelevant information and enhancing useful information to increase image quality. Each pixel per channel is mapped to a value between 0 to 255 to describe the color composition of that pixel mixed from different channels. Wave filters and sharpen operators can be used to restrain noise points and magnify information (especially edge information). Other image augmentation methods such as random flips and random block can also be added to boost variety of dataset. Normalization is used to adjust distribution range of pixel values (e.g. to 0-255 or 0-1).

While with the development of vision transformer, the image preprocessing pipeline can also be words-like. In that context, a image is split into several small patches, which are then fed into a MLP network to embedded vectors (along with class vector of each patch).

2.3. Differences

2.3.1. Data-dependent V.S. Rule-based

In text tokenization process, the tokenizer needs to go through whole whole corpus to construct vocabulary attached to that specific corpus. The mapping rules from tokens to IDs are dependent on dataset itself and different in different corpus; that's why we need to use specific pretrained tokenizer according to the dataset used.

While in image preprocessing all are conducted based on fixed rules. For example, the mapping rules from pixels to values are based on the rules of (gray or RGB or RGBA) color composition and independent of dataset.

2.3.2. Look-up table ID V.S. Model input

The output of text tokenization is a matchup relationship between tokens and IDs. The IDs are not input of language model; instead, each ID corresponds to a randomly initialized embedding vector which is input into the model to represent that word. The embedding vectors as a whole form a look-up table where each column is an embedding feature of a token. While the preprocess image (as a tensor) will be directly fed into model.

3. Technical details

Different from RNN model where we input a sentence in multiple time steps, and in t-th step we input the first (t-1) words to predict word at step t. So natually the model can only predict next word based on previous words it has seen. In transformer model, though, the whole sequence is input at a time and simultaneously predict whole sequence (with one time step lag) . At time step t model could easily predict next word by putting more attention on input at time step (t+1) and simply copy it as output at time step t. But that's not possible in real case. To keep it autoregressive, the self-attention module in decoder of transformer must be masked to ensure only time steps in input sequence before current step can been seen to predict current word.

In code, mask is implemented by function *'_generate_square_subsequent_mask'* in *Transformer.py.* A mask matrix is a upper triangular matrix whose elements in the right upper corner are negative infinity and and in the left bottom corner are 0. This mask matrix is added to the output of query matrix multiplied by the transpose of key matrix ($\mathrm{Q} * \mathrm{K}^{\mathrm{T}}$) to make all elements in the right upper corner negative infinity, which means the attention of query at time t will distribute 0 attention weighst at any time step after t (note the masked weight attention matrix needs a Softmax process and

negative infinity then goes to 0). The formula below clearly demonstrates how the mask works:

$$\mathrm{MHMA}(Q, K, V) = \mathrm{Softmax}\left(\frac{Q * K^{T} + M}{\sqrt{d}}\right) * V$$

Where Q, K, V are query, key and value matrix respectively, d is the dimension of single query/key/value vector. M is the mask matrix whose elements are:

$$M_{ij} = \begin{cases} 0, & i > j \\ -\infty, & i \leq j \end{cases}$$

4. Attention Visualization

For visualizing the attention weights, we pick following sentence in test set to demonstrate: ‘He had the support of university to continue running the program the right way. ’ The attention weight is plot below:

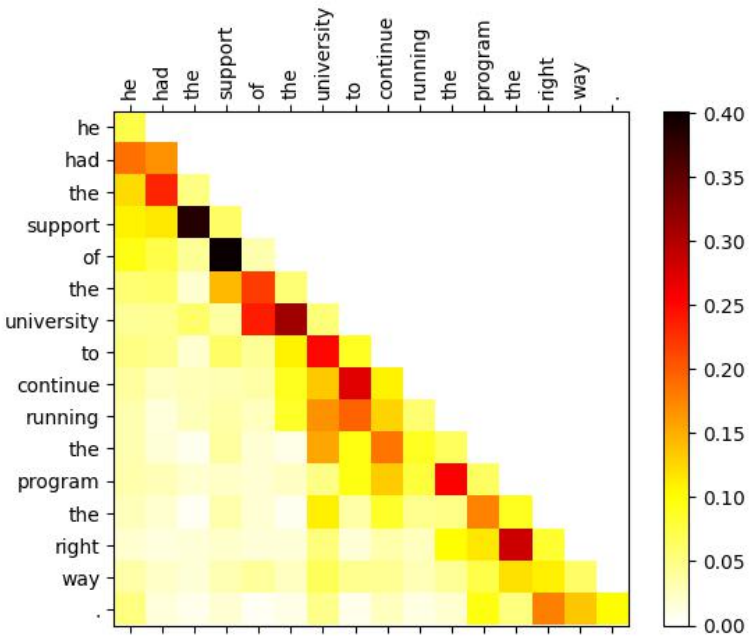

Figure 3 Attention weights on example sequence

Notably, our Transformer model has learned some meaningful weights. For example, the weights of the first ‘the’ querying the subsequent word ‘support’, the second ‘the’ querying ‘university’, the third ‘the’ querying ‘program’ and last ‘the’ querying ‘right’ are all outstanding . Attention of ‘University’ querying ‘continue running’ is also significant, while ‘Right way’ has sort of ability to attend the period ‘ . ’ . But on the other hand we didn’t see the expected long-distance attention, like attention between ‘support’ and ‘program’ which is closely connected in semantic meaning.

5. Extra Techniques

According to training curves, we can see that RNN model has already performed pretty well, while there is a big gap between training and validation loss in Transformer model. Thus we try to use extra techniques to improve the performance of the transformer model.

The focus relies on how to relieve overfitting problem in training set. The big gap in two sets states that some noisy pattern in training set, which dose not contribute to generalization ability, has been learned by Transformer model. Following attempts have been taken to combat with this issue.

In paper *Rethinking the inception architecture for computer vision* (Szegedy, 2016) , a training technique called label smoothing has been raised to polish standard crossentropy loss function. While true label of a classification problem is a hard label in the sense of every single sample has label value 1 in groundtruth class dimension and 0 in other dimensions, label-smoothed crossentropy loss redistributes a small proportion ε to wrong class dimensions averagely and remain 1 - ε for true class. The insight is to slightly punish ‘true classification’ in training set to prevent model from learning all patterns in training set (including general pattern and noises) to achieve maximal classification accuracy on training set only.

Since our index for language model, the perplexity depends on hard label crossentropy loss to compute, when using label-smoothed crossentropy as target function we need to use a seperate crossentropy function to compute perplexity.

With smoothing (ε) = 0.1, we've decreased perplexity to 183.75 in test set (7.9% improving rate). Training and validation curves are as below:

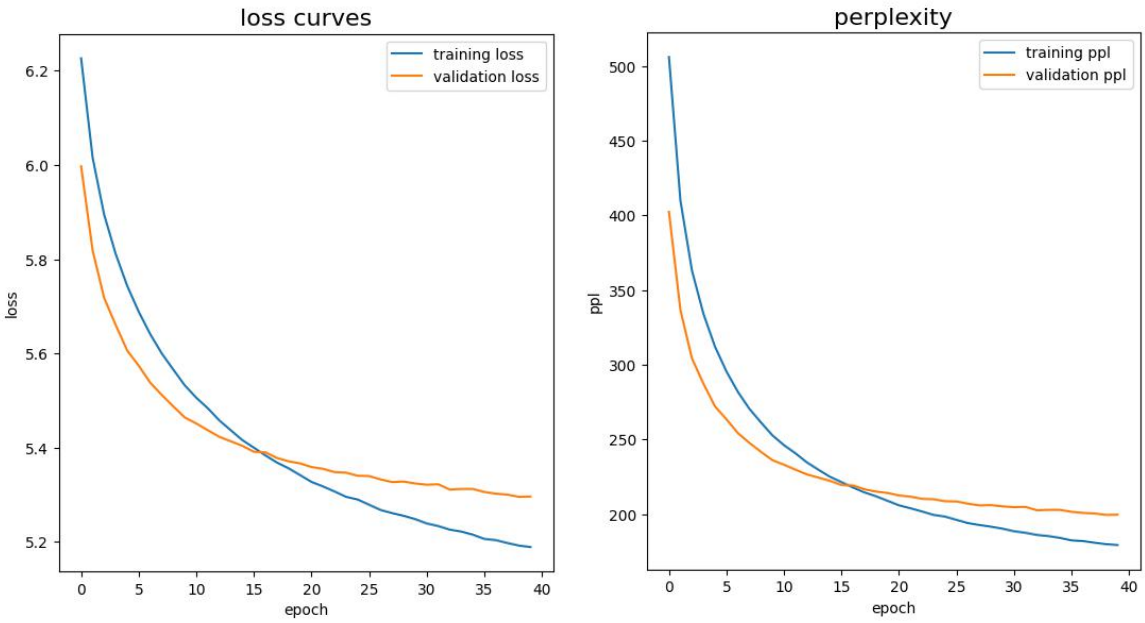

Figure 4 Loss and perplexity curves of Transformer model with smoothed loss

Complete training log can be retrieved in 'screenlog_Transformer_extratech.log' file. Compared to original curves, the generalization gap has been efficiently reduced, meaning our idea of mitigating overfitting is successfully conducted.

References

Szegedy, Christian, V. Vanhoucke, S. Ioffe, Jonathon Shlens and Z. Wojna. *"Rethinking the Inception Architecture for Computer Vision."* 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 2818-2826.

黄民烈，黄斐，朱小燕. 现代自然语言生成.电子工业出版社，76-77