Homework 2: Learning on Sequence Data

Deep Learning (84100343-0)
Autumn 2022
Tsinghua University

1 Background: Language Modeling

Language modeling is a central task in NLP and language models can be found at the heart of speech recognition, machine translation, and many other systems. In this homework, you are required to solve a language modeling problem by implementing RNN and Transformer. A language model is a probability distribution over sequences of words. Given such a sequence $(\mathbf{x}_1, ..., \mathbf{x}_m)$ with length m, it assigns a probability $P(\mathbf{x}_1, ..., \mathbf{x}_m)$ to the whole sequence. In detail, given a vocabulary dictionary of words $(\mathbf{v}_1, ..., \mathbf{v}_m)$ and a sequence of words $(\mathbf{x}_1, ..., \mathbf{x}_t)$, a language model predicts the following word $\mathbf{x}_{t+1}$ by modeling: $P(\mathbf{x}_{t+1} = \mathbf{v}_j | \mathbf{x}_1, ..., \mathbf{x}_t)$ where $\mathbf{v}_j$ is a word in the vocabulary dictionary. Conventionally, we evaluate our language model in terms of perplexity (PP) https://en.wikipedia.org/wiki/Perplexity. Note that, $\mathrm{PP} = \exp(\text{Average Cross Entropy Loss})$.

2 Dataset and Directory Sturcture

The WikiText dataset is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia, which is available under the Creative Commons Attribution-ShareAlike License. The WikiText dataset contains a large vocabulary, where WikiText-2 has a vocabulary of 33,278 and WikiText-103 is over 8 times larger. As it is composed of full articles, the dataset is also suited for models that can capture long-term dependencies.

We use the small version WiKiText-2 dataset, which contains two splits: training and validation set.

- "`./src/`" contains the start code related to data processing and training procedure.
- "`./data/`" contains the training set and the validation set.

3 Part One: RNN

Since vanilla RNNs can be tough to train on long sequences due to vanishing and exploding gradients caused by repeated matrix multiplication, a common variant is the Long-Short Term Memory (LSTM[3]) RNN. LSTMs solve this problem by replacing the simple update rule of the vanilla RNN with a gating mechanism as follows:

Similar to the vanilla RNN, at each timestep we receive an input $x_t \in \mathbb{R}^D$ and the previous hidden state $h_{t-1} \in \mathbb{R}^H$; the LSTM also maintains an H-dimensional cell state, so we also receive the previous cell state $c_{t-1} \in \mathbb{R}^H$. The learnable parameters of the LSTM are an input-to-hidden matrix $W_x \in \mathbb{R}^{4H \times D}$, a hidden-to-hidden matrix $W_h \in \mathbb{R}^{4H \times H}$ and a bias vector $b \in \mathbb{R}^{4H}$.

At each timestep we first compute an activation vector $a \in \mathbb{R}^{4H}$ as $a = W_x x_t + W_h h_{t-1} + b$. We then divide this into four vectors $a_i, a_f, a_o, a_g \in \mathbb{R}^H$ where a_i consists of the first H elements of a, a_f is the next H elements of a, etc. We then compute the input gate $g \in \mathbb{R}^H$, forget gate $f \in \mathbb{R}^H$, output gate $o \in \mathbb{R}^H$ and block input $g \in \mathbb{R}^H$ as:

$$i = \sigma(a_i), f = \sigma(a_f), o = \sigma(a_o), g = \tanh(a_g),$$

where σ is the sigmoid function and $\tanh$ is the hyperbolic tangent, both applied elementwise.

Finally we compute the next cell state c_t and next hidden state h_t as:

$$c_t = f \odot c_{t-1} + i \odot g, h_t = o \odot \tanh(c_t).$$

3.1 Tasks and Scoring

You need to finish the following tasks on the given dataset and start code:

1. **RNN for Language Modeling**: Construct a kind of RNN with GRU[1] block and train your model from scratch on the *train.txt*. To make this task focus on how RNN works in language modeling, you can use some existing GRU block implementation (e.g. *torch.nn.GRU*). Then validate your model on the *valid.txt*, and report the training and validation curves. [**10pts**]
2. **LSTM Implementation**: Construct a kind of RNN with LSTM[3] block and train your model from scratch. Note that in this task, usage of an existing LSTM implementation **is not allowed** (e.g. *torch.nn.LSTM*). Validate your model, and report the training and validation curves. [**15pts**]
3. **Question Answering**: In the current setup, our RNN language model produces a word at every timestep as its output. However, an alternate way to pose the problem is to train the network to operate over **characters** (e.g. 'a', 'b', etc.) as opposed to words, so at it every timestep, it receives the previous character as input and tries to predict the next character in the sequence. Can you give one advantage and disadvantage of a language model that uses a character-level RNN? [**5pts**]

4 Part Two: Transformer

When you finish the previous task, you can find that RNNs are incredibly powerful but still often slow to train when the sequence length is long. Since RNNs struggle to encode long-range dependencies (though LSTMs are one way of mitigating the issue), Transformer[7] is proposed to learn long-range dependencies. The paper not only led to famous models like BERT[2] and GPT[6] in the natural language processing community but also an explosion of interest across fields, including computer vision, time series analysis, and so on.

The capability of learning long-range dependencies can mainly be due to the attention mechanism. Recall that attention can be viewed as an operation on a query $q \in \mathbb{R}^d$, a set of value vectors $\{v_1, \ldots, v_n\}, v_i \in \mathbb{R}^d$, and a set of key vectors $\{k_1, \ldots, k_n\}, k_i \in \mathbb{R}^d$, specified as:

$$c = \sum_{i=1}^{n} v_i \alpha_i, \alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)}$$

where α_i are frequently called the "attention weights", and the output $c \in \mathbb{R}^d$ is a correspondingly weighted average over the value vectors.

In Transformers, we perform self-attention, which means that the values, keys and query are derived from the input $X \in \mathbb{R}^{\ell \times d}$, where ℓ is our sequence length. Specifically, we learn parameter matrices $V, K, Q \in \mathbb{R}^{d \times d}$ to map our input X as follows:

$$\begin{aligned} v_i &= V x_i \;\; i \in \{1, \ldots, \ell\} \\ k_i &= K x_i \;\; i \in \{1, \ldots, \ell\} \\ q_i &= Q x_i \;\; i \in \{1, \ldots, \ell\} \end{aligned}$$

In the case of multi-head attention, we learn a parameter matrix for each head, which gives the model more expressivity to attend to different parts of the input. Let h be number of heads, and Y_i be the attention output of head i. Thus we learn individual matrices Q_i, K_i and V_i. To keep our

overall computation the same as the single-headed case, we choose $Q_i \in \mathbb{R}^{d\times d/h}$, $K_i \in \mathbb{R}^{d\times d/h}$ and $V_i \in \mathbb{R}^{d\times d/h}$. Adding in a scaling term $\frac{1}{\sqrt{d/h}}$ to our simple dot-product attention above, we have

$$Y_i = \text{softmax}\left(\frac{(XQ_i)(XK_i)^\top}{\sqrt{d/h}}\right)(XV_i)$$

where $Y_i \in \mathbb{R}^{\ell\times d/h}$, where ℓ is our sequence length.

In standard implementation, we apply dropout to the attention weights (though in practice it could be used at any step):

$$Y_i = \text{dropout}\left(\text{softmax}\left(\frac{(XQ_i)(XK_i)^\top}{\sqrt{d/h}}\right)\right)(XV_i)$$

Finally, the output of the self-attention is a linear transformation of all concatenation of the heads:

$$Y = [Y_1; \ldots; Y_h]A$$

where $A \in \mathbb{R}^{d\times d}$ is the linear transformation and $[Y_1; \ldots; Y_h] \in \mathbb{R}^{\ell\times d}$.

4.1 Tasks and Scoring

You need to finish the following tasks on the given dataset and start code:

1. **Question Answering**: Several key design decisions are made in designing the multi-head attention we introduced above. Explain why the following choices are beneficial: (1) Using multiple attention heads instead of one. (2) Dividing by $\sqrt{d/h}$ before applying the softmax function, where d is the feature dimension and h is the number of heads. [**10pts**]
2. **Multi-head Attention**: Construct the multi-head attention module based on the given equations above. We provide the start code and checking code in "`./src/mha.py`". If you implement it correctly, the relative error should be less than $1e^{-3}$. [**10pts**]
3. **Transformer for Language Modeling**: Construct a Transformer for language modeling. Validate your model, and report training and validation curves. (**Hint**: You can implement this task by using a decoder-only architecture like GPT[6] or combining the encoder of Transformer with causal masking. You can use existing implementations like *torch.nn.Transformer* but make sure to understand how each component functions.) [**20pts**]
4. **Masking**: The language modeling we introduce in Sec 1 is in fact the Causal Language Modeling, where the model is just concerned with the left context. As we have learned in class, masking is used in Transformer to keep it autoregressive. Explain why masking is necessary for our task and how it is implemented in your code. [**10pts**]
5. **Attention Visualization**: Visualize the attention weights of some typical words and check whether the Transformer can learn meaningful attention weights. [**10pts**]

5 Part Three: Improve Your Language Model

Despite substantial progress has been made in language modeling by using RNN and Transformer, there are still several challenging problems to be solved and related research is pouring in.

One of the challenges is model overfitting. As a large number of neural language models have been shown to be prone to overfitting, BERT[2] find it helpful to alleviate it by adding random masking on the input of the model (known as Masked Language Modeling), and adversarial training mechanism [8] is also proposed for effectively regularizing neural language models.

Also, the efficiency of the self-attention mechanism will become the computational bottleneck when applying Transformers to extremely long sequences. To overcome the quadratic computation growth on sequence length, several works aim to reduce self-attention complexity. LogTrans[5] introduces the local convolution to Transformer and proposes the LogSparse attention to select

time steps following the exponentially increasing intervals. Informer[9] extends Self-Attention with KL-divergence criterion to select dominant queries. Reformer[4] introduces local-sensitive hashing (LSH) to approximate attention by allocated similar queries.

5.1 Tasks and Scoring

You need to finish the following tasks on the given dataset and start code:

- Start with how you feel after completing the first few tasks and adopt an extra technique you find in other materials to further improve your language model (no matter for performance or efficiency). Please explain why you choose it, and check whether it works on our task. [**10pts**]

6 Submit Format

Submit your *code and report* as an Archive (zip or tar). The report is supposed to cover your **question answering**, the model **technical details**, **experimental results**, and necessary **references**.

References

[1] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS*, 2015.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[4] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[5] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[6] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[8] D. Wang, C. Gong, and Q. Liu. Improving neural language modeling via adversarial training. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 6555–6565, 2019.

[9] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.