1. Training a Neural ODE model in Transformer

We discuss the details of training the dynamical model $\boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)$, recall in our FLOATER model, function $\boldsymbol{h}$ joins in the computational graph implicitly by generating a sequence of position encoding vectors $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \dots, \boldsymbol{p}_N\}$, conditioning on a freely initialized vector $\boldsymbol{p}_0$. The generation steps are computed iteratively as follows (suppose we choose the interval between two consecutive tokens to be Δ)

$$\begin{aligned}
\boldsymbol{p}_1 &= \boldsymbol{p}_0 + \int_0^{\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau, \\
\boldsymbol{p}_2 &= \boldsymbol{p}_1 + \int_{\Delta}^{2\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau, \\
&\vdots \\
\boldsymbol{p}_N &= \boldsymbol{p}_{N-1} + \int_{(N-1)\Delta}^{N\Delta} \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau.
\end{aligned} \tag{1}$$

Finally, the loss L of this sequence is going to be a function of all position encoding results $L = L(\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_N)$, which is further a function of model parameters $\boldsymbol{w}_h$. The question is how to calculate the gradient $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h}$ through backpropagation. This question is fully solved in Neural ODE method (Chen et al., 2018b) with an efficient adjoint ODE solver. To illustrate the principle, we draw a diagram showing the forward and backward propagation in Figure 1.

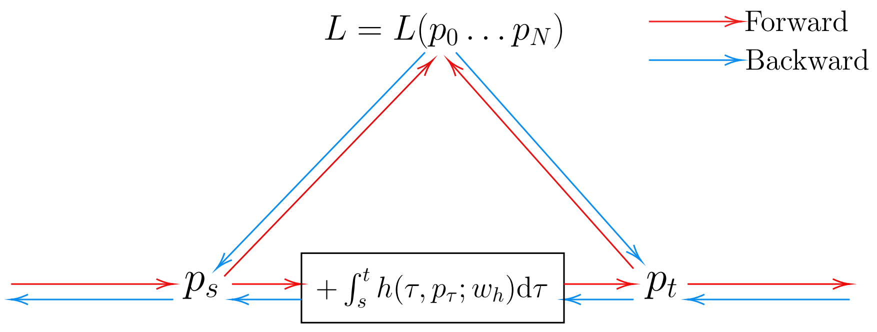

Figure 1. Direction of forward and backward propagation. Here we consider a simplified version where only position encodings $\boldsymbol{p}_s$ and $\boldsymbol{p}_t$ are in the computational graph.

From (Chen et al., 2018b), we know that the gradients $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}_h} L\Big(\boldsymbol{p}_s + \int_s^t \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)\,\mathrm{d}\tau\Big) \triangleq \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h}$ can be computed by

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{w}_h} = -\int_t^s \boldsymbol{a}(\tau)^\intercal \frac{\partial \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h}\,\mathrm{d}\tau, \tag{2}$$

where $\boldsymbol{a}(\tau)$ defined in $\tau \in [s, t]$ is called the "adjoint state" of ODE, which can be computed by solving another ODE

$$\frac{\mathrm{d}\boldsymbol{a}(\tau)}{\mathrm{d}\tau} = -\boldsymbol{a}(\tau)^\intercal \frac{\partial \boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)}{\partial \boldsymbol{p}_\tau}. \tag{3}$$

Note that the computation of (3) only involves Jacobian-vector product so it can be efficiently calculated by automatic differentiation.

2. Implementation details

2.1. Settings of ODE solver

To setup the ODE server, we need to first choose the numerical algorithms (Press et al., 1992). We have different setups for different datasets. For neural machine translation problems (WMT14 En-De and En-Fr), we use the more accurate Runge-Kutta scheme with discretization step $\frac{\Delta}{5.0}$ to solve the adjoint equation (recall that we set the interval of two neighboring tokens to be $\Delta = 0.1$ globally). While for datasets with long sentences such as GLUE and RACE benchmarks, we found that solving the adjoint equation with high order scheme is too slow, in such case we adopt simple midpoint method with discretization step $\frac{\Delta}{5.0}$, and the gradients are calculated by automatic differentiation rather than adjoint method. The third party implementation of ODE solver can be found at `https://github.com/rtqichen/torchdiffeq`.

2.2. Training NMT tasks

We run the same preprocessing script provided by fairseq (Ott et al., 2019), which is also used in ScalingNMT (Ott et al., 2018). With the standard training script, we first successfully reproduce all the results in Transformer paper (Vaswani et al., 2017). Based on that we execute the following protocol to get our results:

1. Train the original Transformer model for 30 epochs.
2. Random initialize FLOATER model of same shape configuration.
3. Copy tensors from the best performing checkpoint (validation set) to initialize FLOATER model. Initialize weights in the dynamical model with small values.
4. Half the peak learning rate (e.g. in Transformer-base + En-De, the peak learning rate is changed from 7.0×10^{-4} to 3.5×10^{-4}).
5. With the warm-initialized FLOATER checkpoint, retrain on the same dataset for 10 epochs (En-De) or 1 epoch (En-Fr).
6. Averaging last 5 checkpoints and compute BLEU score on test split.

2.3. Training language understanding tasks

For GLUE/SQuAD/RACE benchmarks, our experiments are all conducted upon RoBERTa, in which both `base` and `large` configurations are available. Due to resource constraint (and to show the compatibility to existing models), we initialize our FLOATER model with pretrained RoBERTa, which is similar to NMT task. However, the weights $\boldsymbol{w}_h$ in dynamic function $\boldsymbol{h}(\tau, \boldsymbol{p}_\tau; \boldsymbol{w}_h)$ are not trained in large corpus, given that GLUE/SQuAD/RACE datasets are too small to train dynamics from scratch, we decided to pretrain $\boldsymbol{h}$ alone in WikiText103 (Merity et al., 2016) data using masked language modeling loss. We have found that when we train $\boldsymbol{w}_h$ alone, it only takes a few hours (2x Titan V100) and one epoch to convergence.

Once having the pretrained FLOATER model, we can run following downstream tasks and compare with RoBERTa under the same setting:

GLUE benchmark consists of eight datasets and each have different hyperparameter settings. For hyperparameters such as learning rate, batch size, training iterations, warm-up iterations, etc., we use the same values recommended by official repository of RoBERTa[1].

SQuAD benchmark. For this benchmark we wrote our own finetuning code because currently there is no official code available. During the implementation process, we mainly refer to the third-party repositories[2]. We are not able to exactly match the official result reported in RoBERTa paper but quite close (~ 0.1 difference in F1). For our FLOATER model, we use the same hyperparameters as RoBERTa.

RACE benchmark. This benchmark has the longest context and sequence length. We follow the official training script[3] and reproduce the result. Similar to other benchmarks, we then repeat the training process using exactly the same training hyperparameters to make a fair comparison. In this benchmark we freeze the weights $\boldsymbol{w}_h$ and only finetune the weights of RoBERTa.

[1]Available at: `https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.glue.md`

[2]Mainly `https://github.com/ecchochan/roberta-squad` and `https://github.com/huggingface/transformers`

[3]`https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.race.md`

3. Cases suitable for non-equidistant discritization

Although our model allows continuous values of s and t in (8), limiting the scope to text modeling tasks, positions are discrete values as $\{0, 1, 2, \dots\}$. Once the continuous version of position representation $\boldsymbol{p}_t$ is obtained, we simply take the discritized $\{\boldsymbol{p}_0, \boldsymbol{p}_\Delta, \boldsymbol{p}_{2\Delta}, \dots, \}$ as the actual values to feed into Transformer model, where Δ is a hyperparameter (*e.g.* $\Delta = 0.1$). By choosing positions t equidistantly, we are implicitly assuming the position signal evolves steadily as we go through each token in a sentence. More generally, the dynamics in (8) can deal with the case in which positions are not integers $0, 1, 2, \dots$ etc., but arbitrary monotone increasing series $t_0 < t_1 < t_2 < \dots$ which may not be equidistant. In appendix, we exemplify this general situation with several widely deployed tasks; we regard this as a interesting future direction. This makes our model particularly suitable for following scenarios yet traditional position representation may not be good at:

- **Hierarchical Transformer model** (Liu & Lapata, 2019; Zhang et al., 2019). The model is a direct extension of hierarchical RNN and is often used in long document processing. It works by first running a word-level Transformer model on each sentence to extract the sentence embedding, and then applying a sentence-level Transformer scanning through each sentence embedding sequentially. We argue that when processing at the sentence level, it could be better to set the increment of position index $t_{i+1} - t_i$ proportional to the length of the i-th sentence. This is because longer sentences tend to carry more information, so $\boldsymbol{p}_{i+1}$ is likely to move farther from $\boldsymbol{p}_i$.
- **Transformer for time-series events**. As measurement time is continuous, time-series data is another scenario when a continuous position makes more sense than a discrete counterpart. More importantly, to predict the future values by modeling historical values observed at irregular time grids, it is better to consider the length of time horizon between two consecutive measures. A successful previous work is the Latent ODE (Chen et al., 2018a), except that they use RNN as the backbone, and they model the hidden states rather than position representations with Neural ODE (because RNN itself provides positional bias).

In this paper, we are not going to explore the more general cases discussed above. Instead, we decided to leave them as interesting future work.

References

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6571–6583. Curran Associates, Inc., 2018a. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018b.

Liu, Y. and Lapata, M. Hierarchical transformers for multi-document summarization. *arXiv preprint arXiv:1905.13164*, 2019.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. Numerical recipes in c++. *The art of scientific computing*, 2:1002, 1992.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Zhang, X., Wei, F., and Zhou, M. HIBERT: document level pre-training of hierarchical bidirectional transformers for document summarization. *CoRR*, abs/1905.06566, 2019. URL http://arxiv.org/abs/1905.06566.