SOTA Attention Mechanism and Activation Functions on XLNet

Stanford CS224N Custom Project

Haozheng Du
Department of Computer Science
Stanford University
bf3magic@stanford.edu
Project Mentor: Lingjue Xie

Abstract

In this project, we re-implemented a XLNet model in PyTorch from scratch, and experimented with SOTA activation functions including GELU and Mish, as well as Attention on Attention (AoA) mechanism. We analyzed the effect of the above techniques on our XLNet model evaluated on its pretraining behavior and then further implemented such techniques on the original XLNet model. We found beneficial effect of Mish on XLNet, and that AoA helped XLNet converge faster under certain circumstances.

1 Introduction

The transformer architecture enabled various state-of-the-art pretraining approaches to achieve amazing results on language understanding tasks including SQuAD 2.0. Among those high performance models, autoregressive (AR) modeling and autoencoding (AE) modeling are the most successful. AR models including GPT-2[1] and Transformer-XL[2] rely on the transformer decoder part and uses attention masks to make sure that each token can only attend to the tokens before. AE models including BERT[3] and XLM[4] rely on the transformer encoder part, and are able to look at all tokens in the attention head. However, we noticed that XLNet[5], a generalized autoregressive pretraining architecture, addressed and fixed major flaws of both AR and AE models to achieve leading results on a wide variety of NLP tasks. Given that XLNet model is relatively new, we decided to re-implemented the core XLNet architecture and explore different methods to improve its performance.

Various SOTA activation functions emerged in recent years to speed up model training. BERT has moved from ReLU to GELU[6] as its activation function, while XLNet is providing GELU as an option in its codebase. In this project, we implemented GELU and Mish[7] as alternative activation functions for XLNet. We then monitored the pretraining behavior of XLNet, and results showed that Mish helps smooth out learning curve of small XLNet models.

In addition, we built Attention on Attention[8] (AoA), an extension to the conventional attention mechanism, on top of the original Two-Stream Self-Attention mechanism used in XLNet. We then pretrained and finetuned the model on sets of data extracted from SQuAD 2.0 to evaluate its effect. We found AoA could potentially improve XLNet performance under certain environment.

2 Related Work

2.1 XLNet

The XLNet paper discusses limitations of AE language modeling such as BERT. To be specific, BERT constructs a corrupted input by randomly setting 15% of tokens in a sequence to [mask], and train the model to predict such token given all other context in the sequence:

$$\max_{\theta} \log p_\theta(x_{masked}|\hat{x}) = \sum_{t=1}^{T} m_t \log p_\theta(x_t|\hat{x})$$

where $\hat{x}$ is the corrupted input constructed from sequence x, and $m_t = 1$ indicates that x_t is masked.

In this case, BERT is able to look at contextual information from both sides of the token to make predictions, which is considered a remarkable advantage over AR models. However, BERT makes the implicit assumption that masked tokens in the sequence are independent from each other, whereas AR models are pretrained without such assumption. Furthermore, by corrupting input sequences, BERT introduces a pretrain-finetune discrepancy to the model because $[mask]$ token does not exist in downstream tasks. XLNet, on the other hand, propose a permutation language modeling objective to overcome the limitations of BERT mentioned above, while maintaining its bi-directional contextual awareness.

2.2 Attention on Attention

Attention on Attention (AoA) module is an extension to the conventional attention mechanism. According to the AoA paper, it helps determine the relevance between the attention result and query. Studies has shown that AoA significantly boosts deep neural net model performance on image captioning task. However, no comprehensive studies on applying AoA to language models was conducted. In this project, we intended to explore the effect of AoA on XLNet.

3 Approach

We implemented the entire XLNet architecture in PyTorch. Core building blocks including the partial prediction training objective, a Transformer-XL[2] backbone and the Two-Stream Self-Attention mechanism were implemented from scratch to create the model. Though we are aware that there is a released TensorFlow implementation of the original XLNet, we did not directly translated the model to PyTorch. Instead, we implemented the partial prediction and the Two-Stream Self-Attention with minimal reference to the original code, followed along the codebased to create the Transformer-XL backbone and only borrowed the preprocessing part including creating input permutations from the codebase. Given the complexity of XLNet, the amount of work to re-implement the model is non-trivial.

3.1 XLNet Architecture

3.1.1 Overview

The language modeling objective in XLNet is as follows:

$$\max_{\theta} \mathbb{E}_{z\sim Z_t}[\sum_{t=1}^{T} \log p_\theta(x_{z_t}|X_{z_{<t}})]$$

where Z_T is the set of all possible permutations of sequence $[1, 2, ..., T]$, and z_t and $z_{<t}$ is the t-th element and the first $t-1$ elements of a permutation $z \in Z_T$, respectively. To be specific, in XLNet we sample a permutation of the original input sequence each time, and maximize the likelihood of target x, looking at all tokens before x. However, this optimization objective introduces a problem that in order to predict the token x_{z_t}, the model should be aware of its original position z_t but not be aware of the content x_{z_t}.

To solve this problem, the Two-Stream Self-Attention mechanism shown in Figure 1 is implemented, where two sets of hidden state representation are implemented: content representation and query representation. These two sets of representations enabled attention to be computed in different settings: whether or not the model is aware of the target content. During implementation, XLNet incorporates the idea of Two-Stream Self-Attention and the framework of Transformer-XL to construct a model with input and output sequence format as: $[A, \text{SEP}, B, \text{SEP}, \text{CLS}]$, where [SEP] and [CLS] are special tokens to separate the two sequences A and B.

Full implementation details can be found in the original XLNet paper. Here we only discuss the core building blocks of XLNet we implemented.

3.1.2 Partial Prediction

We implemented the partial prediction pretraing objective in XLNet, where only the last tokens in a sequence permutation is predicted. A sequence z is split into two subsequences: $z_{\leq c}$ and $z_{>c}$, where c is the cutting point. Then the model objective is:

$$\max_{\theta} \mathbb{E}_{z \sim Z_T}[\log p_\theta(x_{z_{>c}} | x_{z_{\leq c}})] = \mathbb{E}_{z \sim Z_T}[\sum_{t=c+1}^{|z|} \log p_\theta(x_{z_t} | x_{z_{<t}})]$$

which is the log-likelihood of the target subsequence $z_{>c}$ conditioned on the non-target subsequence $z_{\leq c}$.

3.1.3 Two-Stream Self-Attention

We implemented the Two-Stream Self-Attention algorithm introduced by the paper, where two sets of hidden state representation are created: content representation and query representation. These two sets of representations enabled attention to be computed in different settings: whether or not the model is aware of the target content. To be specific:

$$g_{z_t}^{(m)} \longleftarrow Attention(Q = g_{z_t}^{(m-1)}, KV = h_{z_{<t}}^{(m-1)}; \theta)$$
$$h_{z_t}^{(m)} \longleftarrow Attention(Q = g_{z_t}^{(m-1)}, KV = h_{z_{\leq t}}^{(m-1)}; \theta)$$

The content representation h_{z_t} encodes both the context and token x_{z_t} itslef, whereas the query representation g_{z_t} only encodes the contextual information $x_{z_{<t}}$. Visual representation of the attention structure is shown in Figure 1.

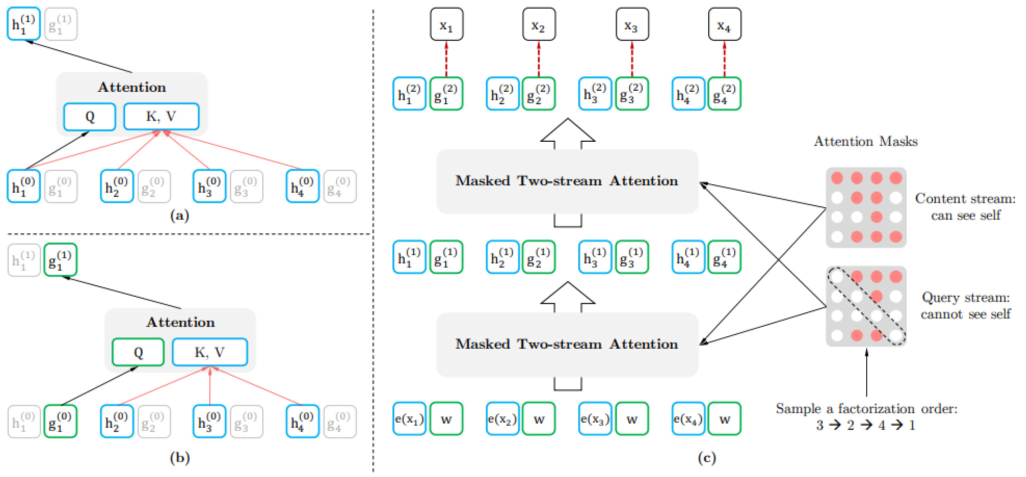

Attention
Q
K, V
(a)
(b)
(c)
Masked Two-stream Attention
Attention Masks
Content stream: can see self
Query stream: cannot see self
Sample a factorization order: 3 → 2 → 4 → 1

Figure 1: Architecture of Two-Stream Self-Attention

3.1.4 Transformer-XL

We followed along the XLNet paper, the Transformer-XL paper, as well as the XLNet codebase to build the model's backbone. The Transformer-XL backbone enabled XLNet better deal with long sentences by using cutting long sequences into segments and passing information from previous segments to current segment. Reference the Transformer-XL paper for full details. In XLNet, two important techniques in Transformer-XL, the relative positional encoding scheme and the segment recurrence mechanism was incorporated. According to the paper, let $\tilde{x} = s_{1:t}, x = s_{T+1:2T}$ be the two segements from a long sequence, and let $\tilde{z}, z$ be the permutation of $\tilde{x}, x$. Now we calculate and cache the content representations $\tilde{h}^{(m)}$ on the segment permutation $\tilde{z}$ for each layer m. Then, for the next segment x, we have:

$$h_{z_t}^{(m)} \longleftarrow \text{Attention}(Q = h_{z_t}^{(m-1)}, KV = [\tilde{h}^{(m-1)}, h_{z_{\leq t}}^{(m-1)}]; \theta)$$

In this way, XLNet can reuse the memory without knowing the factorization order of previous segments. Figure 6 in Appendix shows a figure from the XLNet paper illustrating the architecture.

3.2 Improving XLNet

3.2.1 Attention on Attention

Attention on Attention (AoA) module is an extension to the conventional attention mechanism. It helps determine the relevance between the attention result and query. in AoA, an "information vector" I and an "attention gate" G is generated by applying a linear layer and a non-linear layer respectively on the traditional self-attention result. Then it adds another attention by calculating element-wise matrix multiplication between G and I. The pipeline of AoA is formulated as:

$$G = \sigma(W_q^g Q + W_v^g f_{att}(Q, K, V) + b^g)$$
$$I = W_q^i Q + W_v^i f_{att}(Q, K, V) + b^i$$
$$AoA(f_{att}, Q, K, V) = G \odot I$$

A figure from the AoA paper demonstrating the attention diagram is shown in Figure 2. We built AoA on top of the Two-Stream Self-Attention scores to evaluate its effect on XLNet.

3.2.2 SOTA Activation Functions

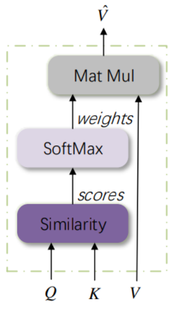
GELU GELU is a high performing activation function proposed by D. Hendrycks et al. in 2016, and it can be viewed as a way to smooth a ReLU. We implemented GELU as an alternative to ReLU in our model. GELU is written as:

$$GELU(x) = xP(X \leq x) = x\Phi(x) = x * \frac{1}{2}[1 + \text{erf}(x/\sqrt{2})]$$

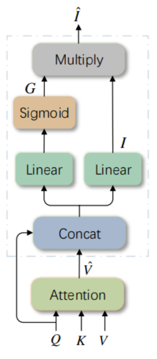
Mish Mish is another novel nerual activation function. Similar to Swish, Mish is a non-monotic activation function defined as:

$$f(x) = x * \tanh(\log(1 + \exp(x)))$$

According to the Mish paper, Mish tends to help deep learning models achieve higher accuracy than a wide variety of different activation functions. Hence, we also implemented Mish as an alternative to ReLU and GELU.



(a) Attention



(b) Attention on Attention

Figure 2: Attention on Attention

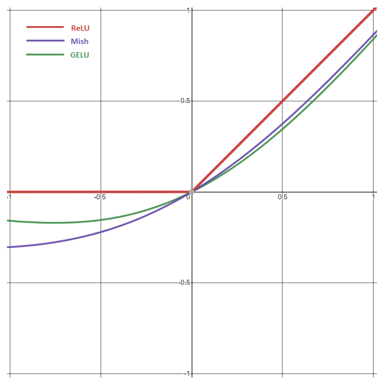


Figure 3: Activation Functions

3.3 Baseline

For the PyTorch model we implemented, we used the model with ReLU activation without AoA as our baseline. We compared the pretraining behavior including learning curve and training loss of the model with GELU/Mish and AoA to the baseline. Then we implemented GELU, Mish and AoA in a larger XLNet model we adapted from the XLNet codebase (implemented in TensorFlow), and conducted comparison and analysis among different model configurations on pretraining and finetuning SQuAD 2.0 dataset.

4 Experiments

4.1 Data

We used SQuAD 2.0 provided in the default project as our dataset, though we were not following the default tracks. The dataset contains many (context, question, answer)triples with a 129,941-6078 train-dev set split. For our PyTorch model, we randomly sampled 10 long context sequences from the training set. We intended to keep the data size very small and pretrain a small model on this dataset. In this way, we can clearly inspect and evaluate the performance difference among different model configurations, without worrying about the noise produced, and the time and resources required for pretraining large models. For the model adapted from the XLNet codebase, we used the entire training set for pretraining and finetuning, in order to answer the question that whether or not the effect of GELU, Mish and AoA we observed on small XLNet would generalize to larger models.

4.2 Data Preprocessing

We adapted the data preprocessing implementation from XLNet codebase. Permutations are created for each text sequence, and are then tokenized with BERT tokenizer (for our PyTorch model) or with the default XLNet tokenizer (for the adapted model). Then an input sequence is prepared as: $[A, \text{SEP}, B, \text{SEP}, \text{CLS}]$ by padding [SEP] and [CLS] tokens to two segments A and B.

For pretraining on the adapted model, we prepared the training set into a txt file where each line is a single sentence. We also separated paragraphs and documents with "<eop>" tokens and empty lines, respectively, since the adapted code has such feature preprocessing already implemented.

4.3 Evaluation method

During pretraining, we used cross entropy loss to monitor the model behavior under different configurations, and evaluated the performance on the loss curve as well as the best (lowest) loss. Then for evaluating performance of the adapted model on SQuAD 2.0, we chose F1 score as well as the loss curve as the metrics. Loss curves were best used to qualitatively evaluate the effect of GELU and Mish, and the final loss and F1 scores were best used to evaluate the effect of AoA.

4.4 Experimental details

4.4.1 Activation Functions

We first experimented with different sets of hyper-parameters on our re-implemented XLNet with ReLU activation function. We tuned the model parameters by pretraining the model on the dataset multiple times to find a good starting point as the baseline. The set of hyper-parameters we chose is shown in Table 3 in Appendix. Then we pretrained the model with GELU and Mish activations with the same set of hyper-parameters on the same dataset. To account for the noise and randomness of learning curve, we pretrained the model 3 times (each time for 1000 epochs) for each activation function used. In this way, we could consolidate our results going forward by observing whether or not the performance difference was reproducible. Every training session (1000 epochs) took around 5 hours on our local machine. We kept record of the loss curve and final loss resulted from each training session for analysis.

Meanwhile, we tuned the hyper-parameters of the adapted XLNet model to fit it on Azure NC12 with two Nvidia K80 GPUs. Though we were using the code directly from the XLNet codebase, we spent plenty of time adopting the model and tuning the model parameters. We observed that it was difficult to get the pretraining loss down, supposedly due to:

- The smaller model size compared to the full XLNet model size.
- Not enough training steps. The full XLNet was pretrained on 512 TPU v3 chips for 500K steps, which took about 5.5 days according to the paper.

Given the limited time and computing power, no extensive experiment was done on picking the best hyper-parameters. We chose a set of parameters shown in Table 4 in Appendix after multiple pretraining attempts. We pretrained the model on the preprocessed SQuAD 2.0 training set described in section 4.2 for about 30 hours. Then we took the pretrained model, and finetuned it on the original SQuAD 2.0 training set. Again, the number of attempts in hyper-parameter tuning was limited because of the time required for every training session. The same set of hyper-parameters was used for ReLU and Mish so that we could compare and analyse the performance difference among them on XLNet.

4.4.2 Attention on Attention

Obtaining preliminary results from Section 4.4.1, we picked the best performing activation function, Mish, and experimented it with AoA. We pretrained our PyTorch model with and without AoA module enabled on the same set of hyper-parameters we used in the previous step, and obtained the pretraining loss curve as well as the final loss.

For the adapted XLNet, we pretrained it on preprocessed SQuAD 2.0 training set with Mish activation and AoA enabled for 30 hours, using the same hyper-parameters we used in 4.4.1. Then we fine-tuned the model on SQuAD 2.0 training set and evaluated it on the dev set. In this way, we could compare the model performance between Mish with AoA and Mish without AoA to analyze the effect of AoA on larger XLNet models.

4.4.3 Remarks on Hyper-parameter Tuning

During our experiment on the adapted XLNet, we found it extremely hard to tune the hyper-parameters to the degree where the model could achieve acceptable performance on the question answering task while having a size small enough to train on NC12 given limited time. Major difficulties include:

- Two sets of parameters needed to be tuned: one for pretraining, the other for fine-tuning.
- More than 10 hyper-parameters needed to be tuned given the complexity of XLNet.
- Training was extremely slow because we had to have small batch sizes to fit training batches in the GPU memory. Therefore, we had limited time to test each hyper-parameter set.

Consequently, we could not conduct comprehensive hyper-parameter tuning to obtain a XLNet model with high performance in the QA task. However, we argue that our major challenges and insight gains come from re-implementing the core building blocks of XLNet, as well as analyzing SOTA activations and novel attention mechanism on the self-implemented model. Therefore, we consider the poor performance of the adapted model acceptable given the scope of this project.

4.5 Results

4.5.1 Self-implemented XLNet

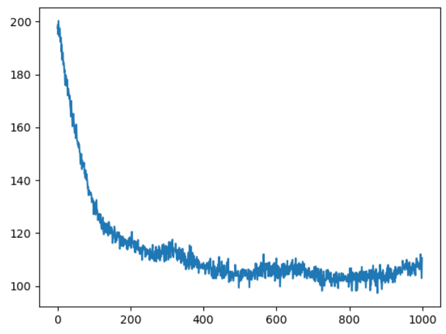
Sample learning curves we obtained from pretraining the re-implemented XLNet are shown in Figure 4. Although the difference between ReLU, GELU and Mish learning curves are small, we could still observe that while ReLU produced the roughest loss curve, Mish produced the smoothest loss curve with fewer bumps and spikes. We also inspected all 9 pretraining learning curves (3 for each activation funcition with AoA disabled), and results showed that the minor performance difference among ReLU, GELU, and Mish was consistent and reproducible.

Comparing Figure 4c and 4d, we observe that Attention on Attention mechanism helps model achieve a much lower loss and smoother learning curve under this hyper-parameter set. We calculated the average of the best losses (3 for each configuration) for each configuration and the results are shown in Table 1. Results show that Mish and ReLU produced a lower pretraining loss than GELU did.

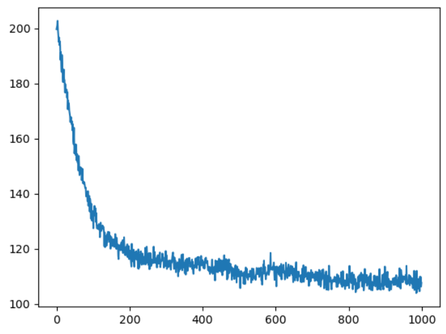
Configuration	ReLU	GELU	Mish	Mish with AoA
Average Loss	102.36	105.02	102.57	66.49

Table 1: Pretraining loss for self-implemented XLNet

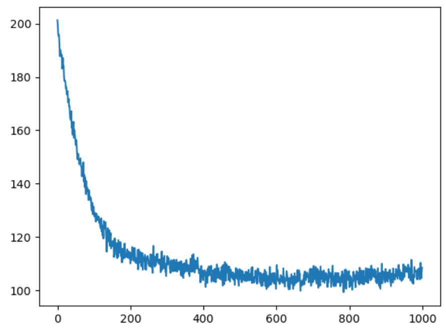
Though ReLU outperforms Mish by 0.21 in loss, we argue that the margin is resulted from noise, and therefore negligible. Furthermore, given the much smoother learning curve of Mish, which could benefit training in practice, we argue that Mish is the best activation function among the three for



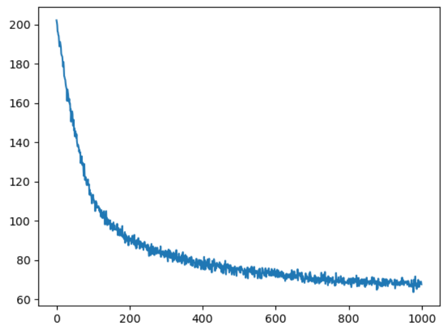
(a) ReLU Baseline



(b) GELU Loss



(c) Mish Loss



(d) Mish with AoA Loss

Figure 4: Learning curve of self-implemented XLNet in pretraining

our XLNet. Combining Mish with AoA yields the best result, but we acknowledge the fact that AoA introduces more trainable parameters to the model, which might result in over-fitting.

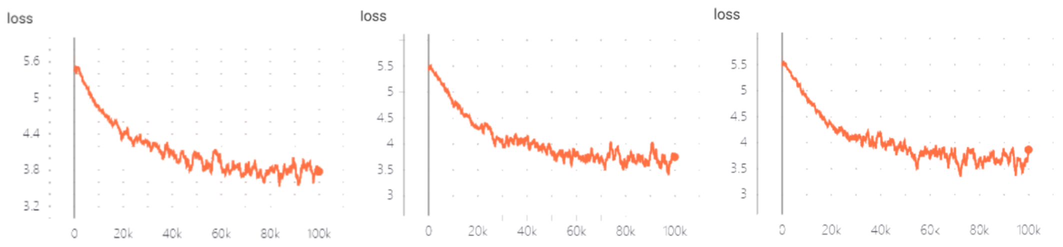
4.5.2 Adopted XLNet Model

Finetuning the adapted XLNet model on SQuAD 2.0 training set, we obtained training loss curves shown in Figure 5. Mish does not clearly outperform ReLU as we saw in pretraining on the self-implemented XLNet. However, we observe that from step 20K to step 60K, where the loss reduction starts to slow down and reach the minimum, learning curve produced by Mish is smoother than the ReLU learning curve. This finding supports, although not in a consolidated way, the previous result that Mish performs better than ReLU. Comparing the learning curve with and without AoA enabled, we do not see significant performance difference.

The dev set F1 scores after finetuning on the training set are shown in Table 2. The poor performance was expected as explained in Section 4.4.3. However, the result is consistent with the result we had in pretraining re-implemented XLNet, in that 1) AoA does help improve the model performance, 2) Mish and ReLU result in similar model performance while Mish helps smooth out the learning process.

Configuration	ReLU	Mish	Mish with AoA
F1	15.36	15.75	18.19

Table 2: F1 scores on SQuAD 2.0 dev set after finetuning



(a) ReLU Loss (b) Mish Loss (c) Mish with AoA Loss

Figure 5: Fine-tuning loss on SQuAD 2.0 training set

5 Analysis

5.1 ReLU, GELU, and Mish

As discussed in Section 4.5.1, we concluded that Mish performs the best in XLNet. Mish smooths out the learning curve by having the following properties over ReLU, which are considered desirable according to previous studies:

- Mish is non-monotonic. This improves expressivity and gradient flow.
- The order of continuity is infinite for Mish. On the other hand, ReLU 0 order of continuity which means ReLU is not continuously differentiable.
- The smoothness of Mish helps with optimization and generalization.

The performance difference among ReLU, GELU, and Mish is clearly shown in pretraining our PyTorch XLNet model because the sequences are fed in one by one, and each sequence is fed into the model 1000 times. Therefore, there is less noise during pretraining and it helps the difference to be expressive. However, we do not see such difference in performance when we fine-tuned the adapted XLNet model, because both the input data and the training task were far more complicated. Nevertheless, the effect of SOTA activations on full-scale XLNet is still to be discussed.

5.1.1 Attention on Attention

From the experiment results we obtained, we concluded that AoA helps improve XLNet's performance. Though the performance advantage of AoA enabled models could partially attribute to 1) models without AoA underfits the data, and 2) models with AoA overfits the data, we still argue that AoA is beneficial for XLNet, especially when computing power is limited. AoA uses simple neural net layers to strengthen the relationship between query and the attention vector, which decreases the training loss by a large margin on small XLNets. To achieve the same performance gain by simply using larger XLNet models would require much more computing power and time. Therefore We recommend implementing AoA mechanism on XLNet when resource is limited. Again, the effect of AoA on full-scale XLNet is yet to be discussed.

6 Conclusion and Future work

In this project, we studied the full XLNet architecture, and implemented its core building blocks from scratch. Given the complexity of XLNet, we referenced the original XLNet codebase to bring these building blocks together to construct our XLNet. Building on top of XLNet, we explored state-of-the-art activation functions including GELU and Mish, and found that Mish is a superior activation function. We justified our findings by pretraining our small, self-implemented XLNet as well as by finetuning the larger, adapted XLNet on SQuAD 2.0. Finally, we studied and implemented AoA on top of the Two-Stream Self-Attention mechanism from XLNet, and found that AoA improves the performance of small-scale XLNet. Given the limited time and computing power, we were unable to implement downstream finetuning tasks on our self-implemented XLNet, and we could not further solidify our primary findings by experimenting on full-scale XLNet. For future studies, we would experiment with full-scale XLNet model with the activation functions and attention mechanism discussed in this project.

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[5] Yiming Yang Jaime Carbonell Ruslan Salakhutdinov Quoc V. Le Zhilin Yang, Zihang Dai. Xlnet: Generalized autoregressive pretraining for language understanding. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[6] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[7] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[8] Lun Huang, Wenmin Wang, Jie Chen, and Xiao-Yong Wei. Attention on attention for image captioning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4634–4643, 2019.

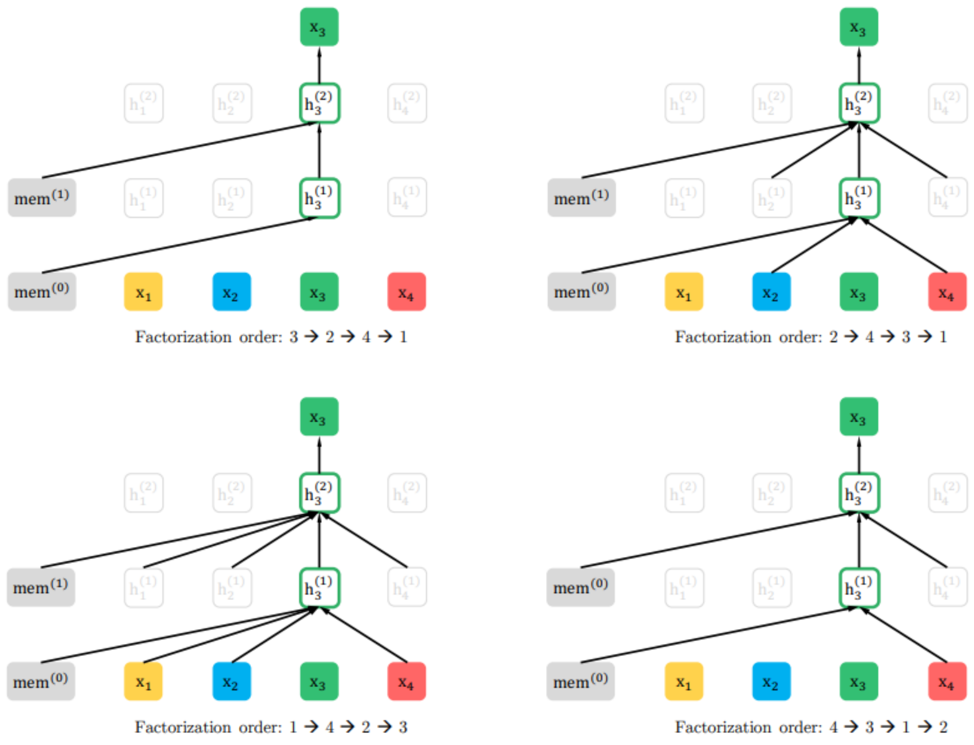
A Appendix

seq_len	512	input sequence length
token_reuse	256	number of tokens used as memory in Transformer XL
mem_len	384	number of steps in memory
alpha	6	number of tokens to form a group
beta	1	number of tokens to mask within a group
num_predict	85	number of tokens to predict
num_epoch	1000	number of epoch
num_layer	6	number of layers
num_head	4	number of attention heads
dim_head	8	dimension of heads
ff_size	32	feed-forward hidden size
batch_size	1	batch size
lr	0.001	learning rate

Table 3: Hyper-parameters for self-implemented XLNet

seq_len	256	input sequence length
token_reuse	128	number of tokens used as memory in Transformer XL
mem_len	128	number of steps in memory
alpha	6	number of tokens to form a group
beta	1	number of tokens to mask within a group
num_predict	40	number of tokens to predict
num_epoch	100	number of epoch
num_layer	12	number of layers
num_head	4	number of attention heads
dim_head	16	dimension of heads
ff_size	512	feed-forward hidden size
pretrain_batch_size	128	batch size for pretraining
pretrain_step	50K	number of steps for pretraining
train_step	100K	number of steps for fine-tuning
train_batch_size	32	batch size for fine-tuning
predict_batch_size	32	batch size for prediction
lr	1e-7	learning rate

Table 4: Hyper-parameters for adapted XLNet



x_3
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
$\mathrm{mem}^{(1)}$ $h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
$\mathrm{mem}^{(0)}$ x_1 x_2 x_3 x_4
Factorization order: 3 → 2 → 4 → 1
Factorization order: 2 → 4 → 3 → 1
Factorization order: 1 → 4 → 2 → 3
Factorization order: 4 → 3 → 1 → 2

Figure 6: Incorporating Transformer-XL