MT-XLN: Multi-Task XLNet for Natural Language Understanding

Practical Deep Learning for system performance : Project Report

Ryan Glassman (rmg2203)

Kavita Anant (ka2744)

Columbia University

***Abstract*—For our final project, we present MT-XLN: Multi-Task XLNet for Natural Language Understanding. The goal is to build on progressions in the state of the art in natural language understanding achieved by XLNet (2019) and MT-DNN (2019). Our approach, in brief, was to alter the architecture of MT-DNN to use XLNet as its shared language model rather than BERT, which was used in the original paper. We hypothesized that ideally, our approach[1] could yield new state-of-the-art results on some benchmark NLU tasks. Even if that was not achieved, we would still get an opportunity to learn more about the behaviors of both XLNet and MT-DNN under different problem settings.**

I. MOTIVATION

Both XLNet [2] and MT-DNN [1] advanced the state of the art on several benchmark NLU tasks when they were published, dethroning the existing SOTA model architecture, BERT. They did so using very different innovations. XLNet defined a new pretraining objective, and MT-DNN introduced a two-phase training procedure that incorporated multi-task learning. Our logic was simple: MT-DNN uses BERT for its lower, shared layers, and XLNet was shown to outperform BERT on most benchmark tasks, including many used in the MT-DNN paper for multi-task learning. By replacing MT-DNN's shared BERT structure with XLNet, could we stack the performance gains of these two methods?

II. BACKGROUND

A. AR Modeling and Autoencoding

Broadly, language modelling can be broken down into two macro approaches: autoregressive (AR) modelling and autoencoding. An AR model, given a text sequence, factorizes the likelihood of the sequence into a forward or backward product. Then a parametric model is trained to model each conditional distribution in the product. The Transformer architecture is a notable example of an AR model. An autoencoder attempts to reconstruct the original data from a corrupted input. Certain tokens in the input are replaced with a special [MASK] token, and the model is trained to recover the masked inputs. BERT (Bidirectional Encoder Representations for Transformers) is a notable example of an autoencoder.

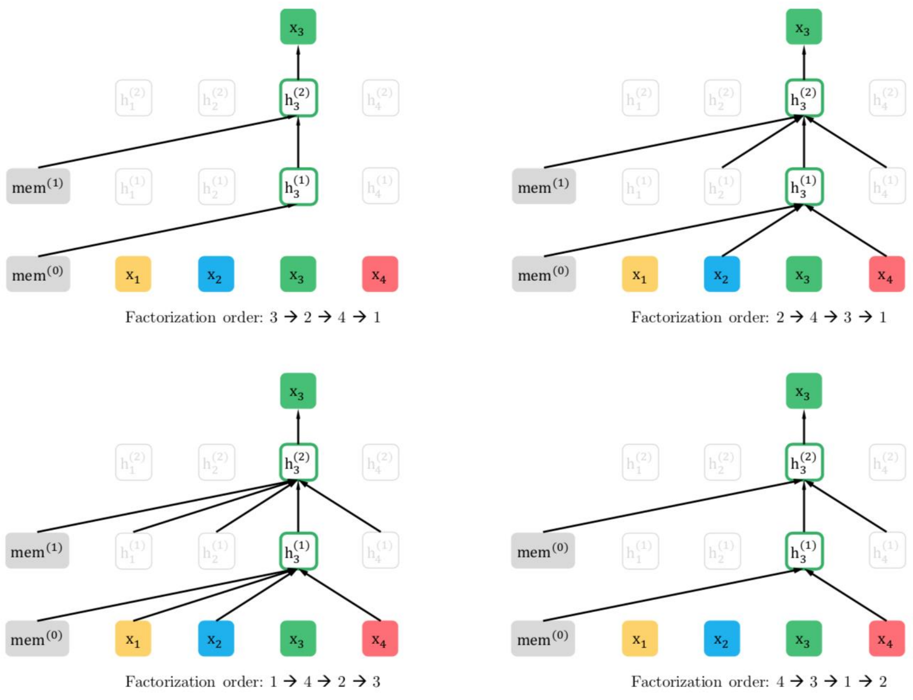
The authors of "Generalized Autoregressive Pretraining for Language Understanding", in which XLNet is proposed, detail some flaws in each approach. AR modelling cannot model bidirectional contexts: decomposing the likelihood of a sequence into a forward or backward product means conditioning the likelihood of a token on either the tokens before it or after it. Whichever direction is chosen, the model loses the ability to learn from the other direction. Autoencoders assume the predicted tokens are independent of each other given the unmasked tokens, which is rarely true in natural language; and by corrupting the training data with [MASK] tokens not present at finetuning, they create a pretrain-finetune discrepancy.

B. XLNet

XLNet aims to create a pretraining objective that capitalizes on the advantages of AR modeling and autoencoding while avoiding each approach's pitfalls. It maximizes the expe[1]cted log likelihood of a sequence with respect to **all possible permutations** of its factorization order. This is still an autoregressive modeling task, so there is no data corruption or assumptions of token independence. But the permutation operation allows the model to capture bidirectional contexts; depending on the factorization order sampled, the context can include tokens on either side of the predicted token.

Below is a figure from the paper showing a diagram of the factorization mechanism. In each diagram, arrows from the x tokens represent contextual information. The sequence factorization is a forward product, so the predicted token (x_3) is conditioned on the tokens before it in the **factorization order**. Therefore, depending on the factorization order, the context for x_3 changes.

[1] https://github.com/glassworks-projects/mtxln



x_3
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$h_4^{(2)}$
$mem^{(1)}$
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_4^{(1)}$
$mem^{(0)}$
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 1: Illustration of the permutation language modeling objective for predicting x3 given the same input sequence x but with different factorization orders.[2]

(The memory modules in grey are inherited from Transformer-XL, from which XLNet gets its name. Explaining them is beyond the scope of our project.)

C. MT-DNN

In 'Multi-Task Deep Neural Networks for Natural Language Understanding,' authors Xiaodong Liu et al, a team of researchers from Microsoft, propose the MT-DNN architecture and the process of multi-task learning. Rather than finetuning a language model on a single task, they construct a multi-headed model with a shared (pretrained) language model foundation, and finetune the full model on multiple tasks. During finetuning, a training batch is sampled from the combined dataset of all training tasks, and the task-specific model head corresponding to the batch task is used for the forward pass. This has two main advantages. First, the procedure leverages supervised data for multiple tasks, which can create a transfer learning effect for tasks with only small amounts of supervised training data. The researchers found that MT-DNN most significantly outperforms previous architectures on such low-data tasks. Second, the network self-regularizes; by training on multiple tasks, the model is prevented from overfitting any one task.

Below is the high-level structure of MT-DNN. In the paper, the authors use BERT for the shared encoder layers.

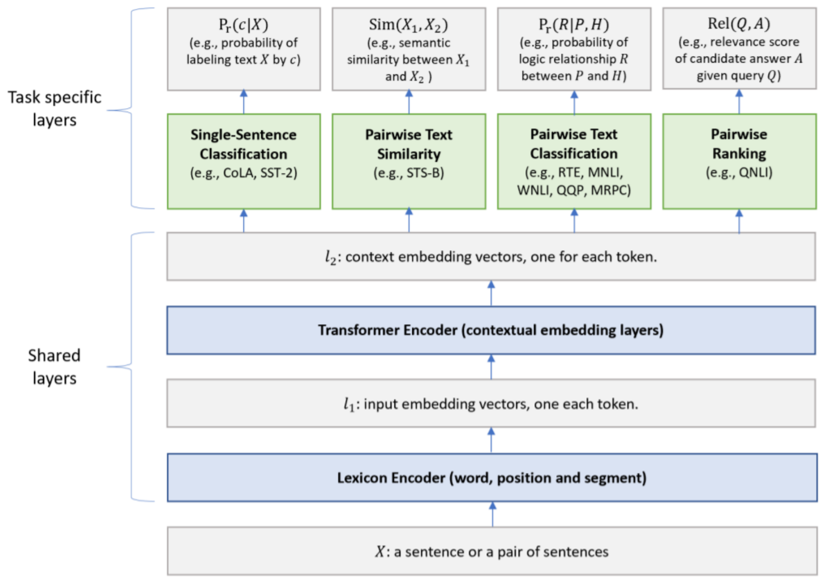

$\mathrm{P_r}(c|X)$
(e.g., probability of labeling text X by c)
$\mathrm{Sim}(X_1, X_2)$
(e.g., semantic similarity between X_1 and X_2)
$\mathrm{P_r}(R|P, H)$
(e.g., probability of logic relationship R between P and H)
$\mathrm{Rel}(Q, A)$
(e.g., relevance score of candidate answer A given query Q)
Task specific layers
Single-Sentence Classification (e.g., CoLA, SST-2)
Pairwise Text Similarity (e.g., STS-B)
Pairwise Text Classification (e.g., RTE, MNLI, WNLI, QQP, MRPC)
Pairwise Ranking (e.g., QNLI)
l_2: context embedding vectors, one for each token.
Transformer Encoder (contextual embedding layers)
Shared layers
l_1: input embedding vectors, one each token.
Lexicon Encoder (word, position and segment)
X: a sentence or a pair of sentences

Figure 2:MT-DNN architecture as suggested in [1]

III. Technical Challenges

When approaching this project, a couple of technical challenges became immediately apparent. The first was simply adapting the MT-DNN architecture to accommodate a different language model structure. Complicating this matter was the fact that the original XLNet code[3] was written in TensorFlow 1.13, and the original MT-DNN code[7] was written in PyTorch. Fortunately, upon further research, we found a number of resources to alleviate these challenges. With their Transformers[4] library, Hugging Face provides open-source implementations of most of the major state-of-the-art language models—both base and with task-specific heads—including BERT and XLNet, along with their corresponding tokenizers, in both TensorFlow 2 and PyTorch. We also found a resource[5] implementing multi-task learning using Hugging Face libraries, written by one of the developers behind Jiant[6]. Adapting this work to support the configurations we wanted to test was thankfully more straightforward than working with the research code directly.

Model size, data size, and hardware availability presented a significant challenge as well. In the paper, XLNet-large (the full 24-layer model) was pretrained on 512 TPUs. Matching that memory size with GPUs would require one GPU per training sample for an input sequence length of 512 (the setting in the paper). In other words, training with a batch size of 32 would require 32 GPUs, each with 16 GB of VRAM. Similarly, MT-DNN was finetuned on 8 V100 GPUs in the paper. To properly compare our results with those of the papers, we would need access to a similar setup. Without such access, we would need to compare our architecture some other way.

IV. Approach and Architecture

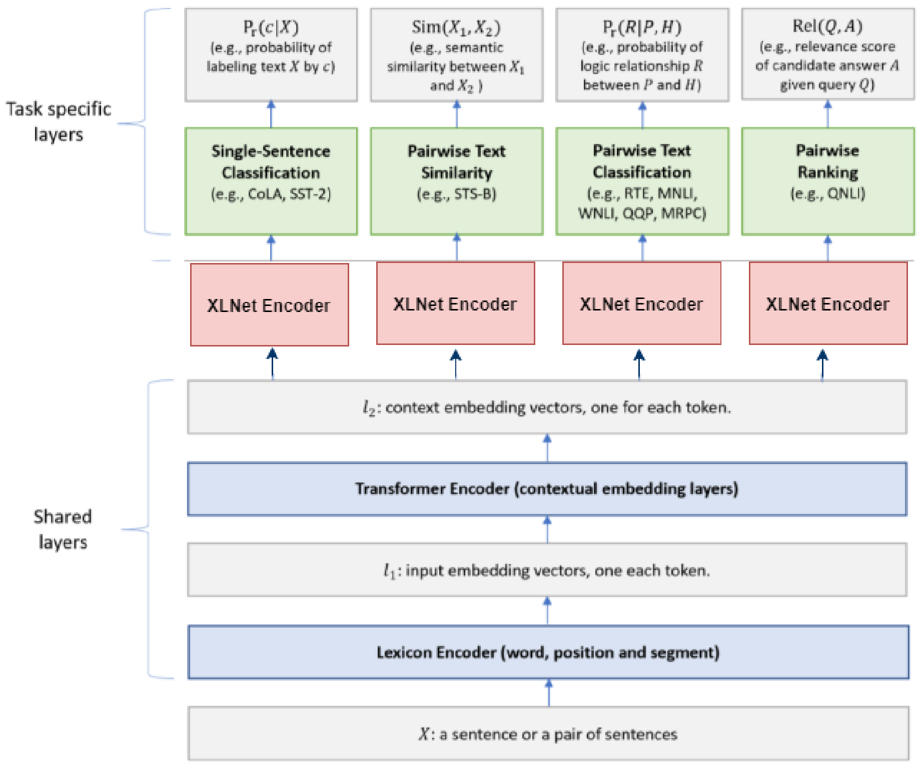

$P_r(c|X)$ (e.g., probability of labeling text X by c)
$Sim(X_1, X_2)$ (e.g., semantic similarity between X_1 and X_2)
$P_r(R|P,H)$ (e.g., probability of logic relationship R between P and H)
$Rel(Q,A)$ (e.g., relevance score of candidate answer A given query Q)
Task specific layers
Single-Sentence Classification (e.g., CoLA, SST-2)
Pairwise Text Similarity (e.g., STS-B)
Pairwise Text Classification (e.g., RTE, MNLI, WNLI, QQP, MRPC)
Pairwise Ranking (e.g., QNLI)
XLNet Encoder
XLNet Encoder
XLNet Encoder
XLNet Encoder
l_2: context embedding vectors, one for each token.
Transformer Encoder (contextual embedding layers)
Shared layers
l_1: input embedding vectors, one each token.
Lexicon Encoder (word, position and segment)
X: a sentence or a pair of sentences

Figure 3: Suggested XLNet MTDNN architecture

HuggingFace library contains a wide range of pretrained language models including XLNet. These models are, however more suitable for single task DNN. In order to make them compatible with the Multitask model, we had to make few changes. Since each of the GLUE tasks [8] have a different kind of input, including a separate XLNet encoder for each task made more sense as suggested in [5]. Rather than using a single shared transformer, use multiple models with shared encoder. We finetune this on our pretrained models and verify the results. We created MultitaskModel class which inherits from transformers.PreTrainedModel and Multitask Trainer class which inherits from transformers.Trainer from the huggingface library that extends the single class functions to a Multitask setting.

V. Implementation Details

We used PyTorch as our learning framework. Since our hardware constraints prevented us from comparing our results to those of the reference papers, we instead conducted our experiments for both a multitask model with an XLNet base and one with a BERT base. We used XLNet-base (cased) and BERT-base (cased)—both 12-layer architectures with approximately 110M parameters—as our base language models, and used their corresponding tokenizers. Both language models were loaded pretrained, as doing language model pretraining was well beyond the scope of our abilities for this project. For each multi-task training run, We trained on one V100 GPU for three epochs. We used the Adam optimizer and a learning rate of 5e-5, as in the MT-DNN paper.

Due to our hardware constraints, we used a maximum sequence length of 340 (instead of the default 512) when tokenizing the GLUE datasets, as well as a batch size of 8 during training. To keep training runs to a manageable length, rather than training on all nine glue tasks, we selected three tasks: one single-sentence classification task, one pairwise text similarity task, and one pairwise text classification task. We selected CoLA, STS-B, and WNLI, respectively, for their relatively manageable dataset sizes. Formulating training this way kept training runs below an hour, allowing us to conduct multiple experiments in a relatively short period of time.
We recorded the metrics and training time of both the models for comparison purposes.

VI. Results

We used the GLUE tasks in different combinations in the MT DNN models. The evaluation scores of the same are shown in Table 1. Our model performed better than the BERT MT-DNN model for RTE, STS-B and WNLI tasks. It was observed, however that these scores we comparatively really low for the CoLA and QNLI tasks.

Task	**RTE (Accuracy)**	**QNLI (Accuracy)**	**CoLA (Mathew's Corr)**	**STS-B (Pearson/ Spearman)**	**WNLI (Accuracy)**
BERT	0.47	0.85	0.521	0.867/0.86	0.45
XLNet	**0.52**	0.53	0.258	**0.885/0.88**	**0.56**

Table 1:GLUE task metrics comparison of XLNet MTDNN and BERT MTDNN

Our understanding of this unexpected behaviour of XLNet was that XLNet models do not work well on short input sentences as they are trained on longer sequences. In [2] the maximum size of the input was 512. But due to our memory limitations, we could not go beyond 340. In order to verify the dependence of XLNet's performance on shorter inputs, we trained and evaluated our model for varying maximum length size from 8 to 128.

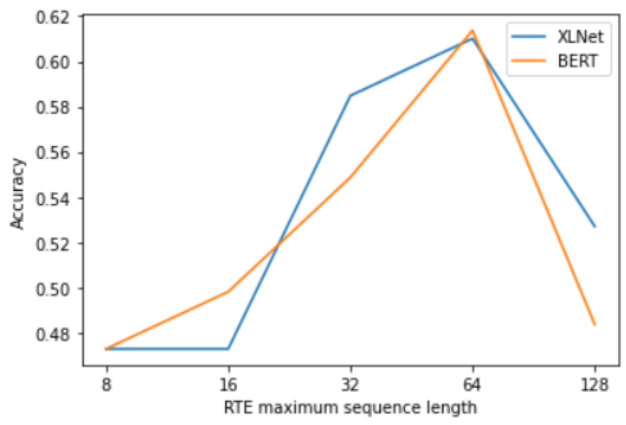


Figure 4: Dependence of Accuracy on maximum length in RTE

Figure 4 and 5 clearly show that the accuracy of the model indeed increases significantly when we increase the maximum length sequence. QNLI and CoLA are larger values as compared to STS-B, WNLI, and RTE. Due to this, when we plot the loss curves of XLNet and BERT in Figure 6 we see that BERT has better loss values as compared to XLNet. Thus, suboptimal performance of larger dataset can skew the data for Multitask learning. To train the models XLNet took 30.44 which almost twice as much as BERT which took 18.56 minutes.

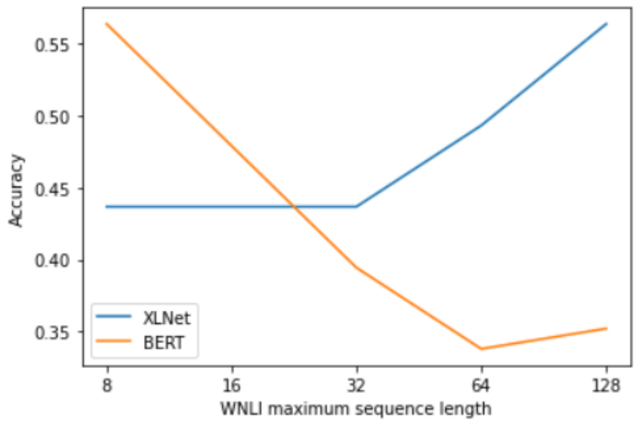


Figure 5: Dependence of Accuracy on maximum length in WNLI

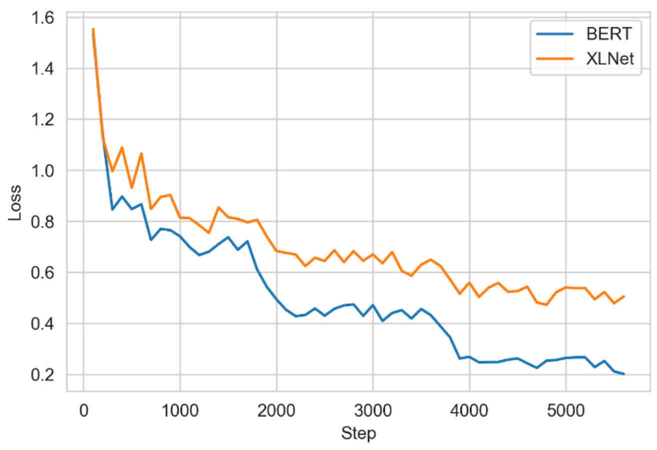


Figure 6: Loss comparison for both the models

VII. Conclusion

XLNet shows significant improvement over BERT as a standalone language model. However, our limited results are inconclusive on the question of whether XLNet outperforms BERT in a multi-task setting. XLNet requires large input sequence lengths to train well, meaning that multi-task learning with BERT may be a better choice in resource-restricted training environments. Additionally, XLNet takes considerably longer to train than BERT—around 67% longer on average, in our tests—which is worth considering depending on the particular application.

There is much work to be done here in the future. Our ability to run multiple tests was limited due to lack of GPU availability—meaning we could run only one test at a time—and, simply, deadlines. Some clear next steps include:

- Training on a multi-GPU setup
- A full GLUE multi-task setup, using all nine GLUE tasks as task heads for multi-task learning
- Conducting an additional round of task-specific finetuning after the multi-task learning stage, as was done in the MT-DNN paper
- Testing XLNet alone (single-task implementation) on the GLUE tasks, and comparing those results to MT-DNN with BERT and MT-XLN
- Conducting a controlled study of different language model bases for multi-task learning
- More robust hyperparameter searches

References

[1] Xiaodong Liu, Pengcheng He, Weizhu Chen, Jianfeng Gao Multi-Task Deep Neural Networks for Natural Language Understanding
[2] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le XLNet: Generalized Autoregressive Pretraining for Language Understanding
[3] https://github.com/zihangdai/xlnet
[4] https://huggingface.co/transformers/
[5]https://colab.research.google.com/github/zphang/zphang.github.io/blob/master/files/notebooks/Multi_task_Training_with_Transformers_NLP.ipynb
[6] https://jiant.info/
[7]https://github.com/namisan/mt-dnn
[8] https://gluebenchmark.com/