

XLNet: Generalized Autoregressive Pretraining for Language Understanding

Abstract: The improvement over the BERT's autoencoding based pretraining model which uses the autoregressive language modeling technique idea with the capability of bidirectional contexts which is an improvement over many state of the art models used before.

REPORT:

Introduction: Autoregressive (AR) language modeling basically predicts the next token given previous tokens. BERT on the other hand does Autoencoding (AE) by masking the tokens in between. The important observation about XLNet is that it captures the dependencies between the predicted tokens also which is omitted in the BERT model which considered the masked tokens as independent entities.

Methodology: The pitfall of AR process is basically that it follows the previous sequence of tokens and have no idea about the next or future tokens for reference. This problem is solved in XLNet as it uses the permutations of ordering to order the tokens randomly and covering all possible permutations so that it can capture the dependencies of all the available tokens in a particular sequence.

$$P(t_1 t_2 t_3) = P(t_1) * P(t_2|t_1) * P(t_3|t_1 t_2)$$

AR language modeling seeks to estimate the probability distribution of a text corpus and given a sequence X , AR modeling factorizes the likelihood into a forward product $p(x) = \text{Product}(t=1 \text{ to } T) p(x_t | x_{<t})$ or a backward product $p(x) = \text{Product}(t=T \text{ to } 1) p(x_t | x_{>t})$. So, basically in AR we can go in one direction only, i.e., either forward or backward but in XLNet we define a concept of finding all possible ordering. Now, by different ordering by sampling, we can get a context of both backward and forward using a particular ordering.

XLNet vs BERT: Given a sequence of text $X = [X_1, X_2, \dots, X_t]$, AR modelling performs pre-training by maximizing the likelihood under the forward autoregressive factorization according to the following equation:

$$\max_{\theta} \log p_{\theta}(x) = \sum_{t=1}^T \log p_{\theta}(x_t | x_{<t}) = \sum_{t=1}^T \log \frac{\exp(h_{\theta}(x_{1:t-1})^{\top} e(x_t))}{\sum_{x'} \exp(h_{\theta}(x_{1:t-1})^{\top} e(x'))},$$

Where $h(x_{1:t-1})$ is a context representation produced by neural models (RNNs or Transformers), and $e(x)$ denotes the embedding of x .

BERT works in autoencoding modelling according to the following equation:

$$\max_{\theta} \log p_{\theta}(\bar{x} | \hat{x}) \approx \sum_{t=1}^T m_t \log p_{\theta}(x_t | \hat{x}) = \sum_{t=1}^T m_t \log \frac{\exp(H_{\theta}(\hat{x})_t^{\top} e(x_t))}{\sum_{x'} \exp(H_{\theta}(\hat{x})_t^{\top} e(x'))},$$

Where $m_t=1$ indicates X_t is masked, and H is a transformer that maps a length- T text sequence X into a sequence of hidden vectors. \hat{x} are the unmasked sequences and it approximately factorizes the log probability of $p(\theta)$.

Takeaway: BERT factorizes (approximately) the joint conditional probability $P(X_- | X^{\wedge})$ based on an independence assumption that all masked tokens X_- are separately reconstructed. In comparison, the AR language modelling objective is to factorize $P(X)$ using the product rule that holds universally without such an independence assumption.

Ordering is very crucial in order to get all the required permutations of all the tokens. In order to manage this, we need to pass the previous tokens to be predicted to the next tokens to be predicted as an input so we use an

Attention technique known as **Masked Two-Stream Attention** which finds dependencies between multiple orderings of permutations and contains two hidden states **h** and **g** in a single layer. One is known as **content stream** which can see self and the other one is **query stream** which cannot see self. There are some more changes in the models like introduction of Transformers-XL which helps increase the accuracy and also different embedding techniques are also being used. It is having multiple hyperparameters which can be tuned to get the results of a particular application in the field of NLP. These are:

Number of layers – 24, Hidden size – 1024, Attention Heads – 16, Attention head size – 64, FFN inner hidden size – 4096, Dropout – 0.1, Attention dropout – 0.1, Partial prediction K – 6, Max sequence length – 512, Memory length – 384, Batch size – 2048, learning rate – 1e-5, Number of steps – 500K, Warmup steps – 20,000, Learning rate decay – linear, Adam epsilon – 1e-6, Weight decay – 0.01.

Results:

We ran the pre-trained model on IMDB dataset by doing some fine tuning to get the following results:

```
020-02-13 13:05:21.812393: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
020-02-13 13:05:21.813599: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1159] Device interconnect StreamExecutor with strength 1 edge matrix:
020-02-13 13:05:21.813622: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1165] 0
020-02-13 13:05:21.813632: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1178] 0:  N
020-02-13 13:05:21.813732: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1), but there must b
020-02-13 13:05:21.814266: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1), but there must b
020-02-13 13:05:21.814860: W tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH enviro
020-02-13 13:05:21.814901: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1304] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1412
NFO:tensorflow:Running local_init_op.
0213 13:05:43.353467 139761444861824 session_manager.py:500] Running local_init_op.
NFO:tensorflow:Done running local_init_op.
0213 13:05:43.776070 139761444861824 session_manager.py:502] Done running local_init_op.
NFO:tensorflow:Saving checkpoints for 0 into exp/imdb/model.ckpt.
0213 13:05:55.591922 139761444861824 basic_session_run_hooks.py:606] Saving checkpoints for 0 into exp/imdb/model.ckpt.
NFO:tensorflow:exp/imdb/model.ckpt-0 is not in all_model_checkpoint_paths. Manually adding it.
0213 13:07:21.580155 139761444861824 checkpoint_management.py:103] exp/imdb/model.ckpt-0 is not in all_model_checkpoint_paths. Manually adding it.
020-02-13 13:07:34.693561: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
NFO:tensorflow:loss = 0.83392364, step = 0
0213 13:07:37.356140 139761444861824 basic_session_run_hooks.py:262] loss = 0.83392364, step = 0
NFO:tensorflow:global_step/sec: 0.90342
0213 13:09:28.046200 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.90342
NFO:tensorflow:loss = 0.73736227, step = 100 (110.691 sec)
0213 13:09:28.047410 139761444861824 basic_session_run_hooks.py:260] loss = 0.73736227, step = 100 (110.691 sec)
```

```
NFO:tensorflow:exp/imdb/model.ckpt-3000 is not in all_model_checkpoint_paths. Manually adding it.
0213 14:06:11.008624 139761444861824 checkpoint_management.py:95] exp/imdb/model.ckpt-3000 is not in all_model_checkpoint_paths. Manually adding it.
NFO:tensorflow:global_step/sec: 0.545769
0213 14:06:13.910520 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.545769
NFO:tensorflow:loss = 0.30387086, step = 3000 (183.228 sec)
0213 14:06:13.911674 139761444861824 basic_session_run_hooks.py:260] loss = 0.30387086, step = 3000 (183.228 sec)
NFO:tensorflow:global_step/sec: 0.993079
0213 14:07:54.607438 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.993079
NFO:tensorflow:loss = 0.75812244, step = 3100 (100.697 sec)
0213 14:07:54.608451 139761444861824 basic_session_run_hooks.py:260] loss = 0.75812244, step = 3100 (100.697 sec)
NFO:tensorflow:global_step/sec: 0.998831
0213 14:09:34.724476 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.998831
NFO:tensorflow:loss = 0.0019574503, step = 3200 (100.117 sec)
0213 14:09:34.725450 139761444861824 basic_session_run_hooks.py:260] loss = 0.0019574503, step = 3200 (100.117 sec)
NFO:tensorflow:global_step/sec: 0.996361
0213 14:11:15.089716 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.996361
NFO:tensorflow:loss = 0.016890004, step = 3300 (100.366 sec)
0213 14:11:15.091084 139761444861824 basic_session_run_hooks.py:260] loss = 0.016890004, step = 3300 (100.366 sec)
NFO:tensorflow:global_step/sec: 0.999415
0213 14:12:55.149315 139761444861824 basic_session_run_hooks.py:692] global_step/sec: 0.999415
NFO:tensorflow:loss = 0.0054993634, step = 3400 (100.058 sec)
0213 14:12:55.149372 139761444861824 basic_session_run_hooks.py:260] loss = 0.0054993634, step = 3400 (100.058 sec)
NFO:tensorflow:Saving checkpoints for 3300 into exp/imdb/model.ckpt.
```

Conclusion:

It is clearly visible that **XLNet** is performing better than **BERT** and other state of the art models. Though a huge question arises as how much the results of **XLNet** is affected by the **Transformer-XL** base model used. It can be tested in other tasks such as **reinforcement learning**. Also, there was an absence of **permutation-based training** and also the performance can be checked when we use abstraction layers such as **Spacy**. Though further applications and improvements are needed to be discussed. We would like to work on it more and in some algorithmic or application based and currently going through the implementation in **PyTorch**.