# 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(t1 t2 t3) = P(t1)*P(t2|t1)*P(t3|t1 t2)**

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) = Product (t=1 to T) p($x_t$ | $x_{<t}$)** or a backward product **p(x) = Product (t=T 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 = [X1, X2, ……, Xt], AR modelling performs pre-training by maximizing the likelihood under the forward autoregressive factorization according to the following equation:

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)},$$

Where h(X1: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:

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

Where mt=1 indicates Xt is masked, and H is a transformer that maps a length-T text sequence X into a sequence of hidden vectors. 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^) 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.

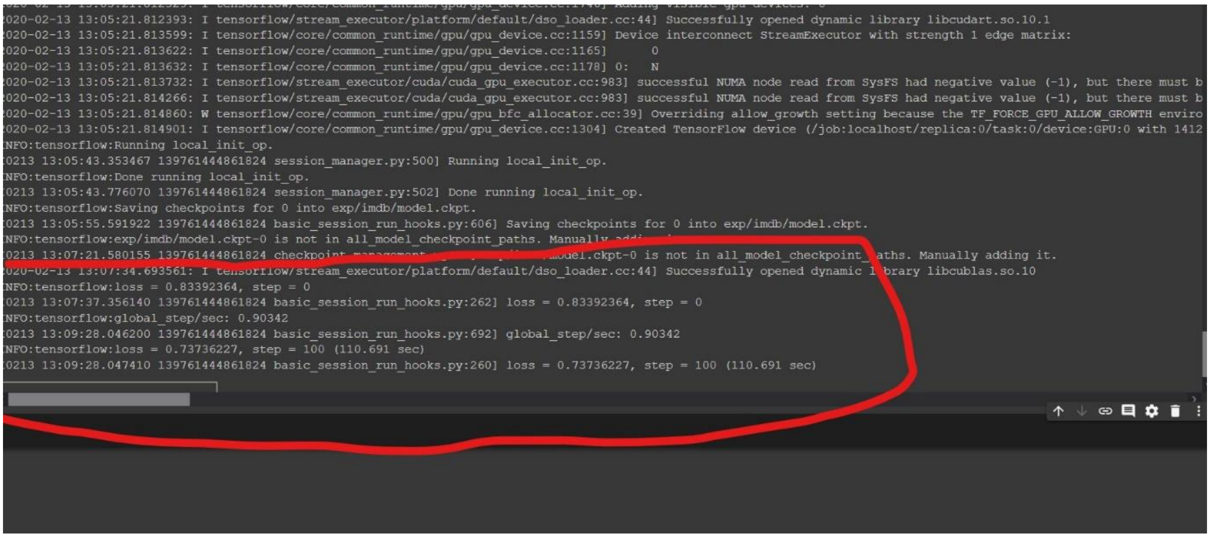
Ordering is very crucial in order to get all the required permutations of all the tokens. In order to manage this, we need to pass the previous tokens to be predicted to the next tokens to be predicted as an input so we use an

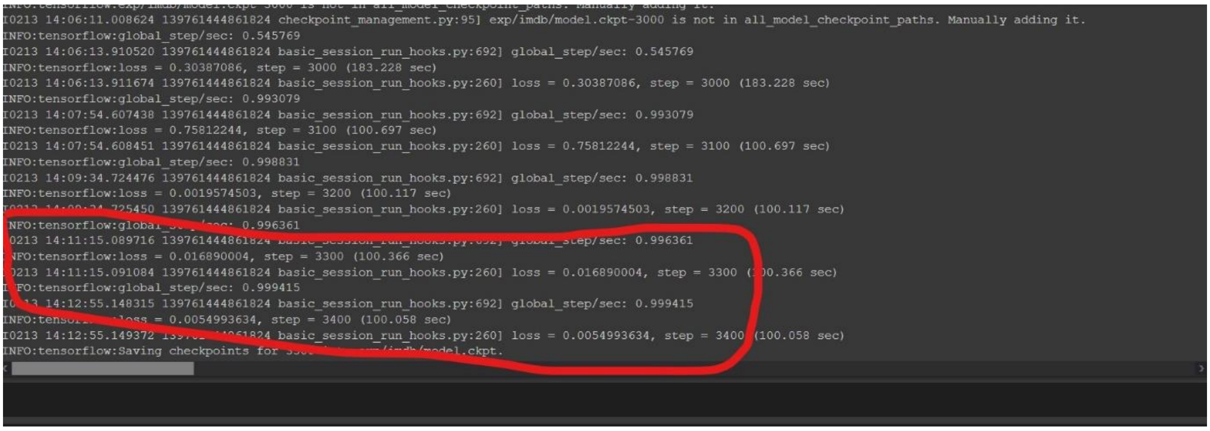
**Attention** technique known as **Masked Two-Stream Attention** which finds dependencies between multiple orderings of permutations and contains two hidden states **h** and **g** in a single layer. One is known as **content stream** which can see self and the other one is **query stream** which cannot 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:




# 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**.