A COMPARISION OF AUTO REGRESSIVE & NON-AUTO REGRESSIVE APPROACHES USING TRANSFORMER MODEL FOR MACHINE TRANSLATION TASK

Javed Roshan, Gabriel Ohaike
W266: Natural Language Processing with Deep Learning
University of California, Berkeley School of Information
{jroshan, gohaike}@berkeley.edu
April 2021

ABSTRACT

Auto Regressive (AR) models use tokens generated in previous time-step as input to calculate outputs. The Transformers are by-and-large autoregressive in nature. Coupled with an attention mechanism, an auto regressive transformer model gets the right context thereby increasing its accuracy. In contrast, Non-Auto Regressive (NAR) models generate a sequence of tokens in parallel removing the reliance on the tokens from previous time-steps. This approach significantly reduces the inference latency of the output. However, at the expense of low accuracy. In this project, we explore the implementations of both these approaches and assess options to narrow the accuracy gap. We changed the architecture of the decoder as part of the NAR implementation and introduced Conditional Random Fields (CRF) to generate the output sequence of a machine translation task. We used IWSLT dataset for German (DE) to English (EN) translation task. We were able to train both the models from scratch on a GPU based server and we observed BLEU scores of AR & NAR to be 16.07 and 8.79 respectively.

1. Introduction

Transformer models were introduced by Vaswani et al[3]. In the context of machine translation, these models are inherently autoregressive. They have an encoder-decoder architecture and they have high inference latency. The latency can potentially inhibit the adoption of these models in prime-time applications. Given a source sentence as a sequence of tokens, $x = (x_1, x_2, \ldots, x_n)$, and a target sentence as another sequence of tokens, $y = (y_1, y_2, \ldots, y_m)$, autoregressive sequence models get from source to target based on chain of conditional probabilities. It has a left-to-right causal structure as follows:

$$\mathrm{p}(y|x) = \Pi_{\mathrm{i}=1}^{\mathrm{m}}\, p(y_i|\, y_{<i}, x) \qquad (1)$$

where $y_{<i}$ represents the tokens before the ith token of target y.

A typical inference cycle is implemented in a conditional loop until an end-of-sentence token is encountered. Every token generated in this loop is dependent on the token from the previous step and a sequence of all generated tokens are provided as input in the current step. The inference process is started with a beginning-of-sentence token. Figure 1a. highlights this process.

Similar to the autoregressive model, non-autoregressive models also follow encoder-decoder architecture. Given a source sentence, $x = (x_1, x_2, \ldots, x_n)$, the encoder takes it as an input and predicts a target sentence's length, m[1]. It also generates source sentence's contextual embedding. The decoder uses a combination of well-designed deterministic input, z, and the encoder created contextual embedding to predict the target tokens

$$\mathrm{p}(y|x) = p(m|x)\, p(z|x)\, \Pi_{\mathrm{i}=1}^{\mathrm{m}}\, p(y_i|z,x) \qquad (2)$$

While the NAR models optimize latency, they do suffer in the accuracy of the output as each target token is generated independently. Multimodality is one of the main problems NARs exhibit [2]. This problem is specifically tackled by using a structured inference module like Conditional Random Fields (CRF) – which model better structural dependencies [2].

$$\mathrm{p}(y|x) = p(m|x)\, p(z|x)\, .\, softmax\left(\Sigma_{i=2}^{\mathrm{m}}\, \theta_{i-1,i}(y_{i-1}, y_i)|\, z, x\right) \qquad (3)$$

where, $\theta_{i-1,i}$ is the pairwise potential for (y_{i-1}, y_i). This probability form could better model the multiple modes in target translations.

2. Architecture

The architecture of an autoregressive and non-autoregressive transformer model is highlighted in Figure 1 and Figure 2[2] respectively. They both have encoder-decoder architecture.

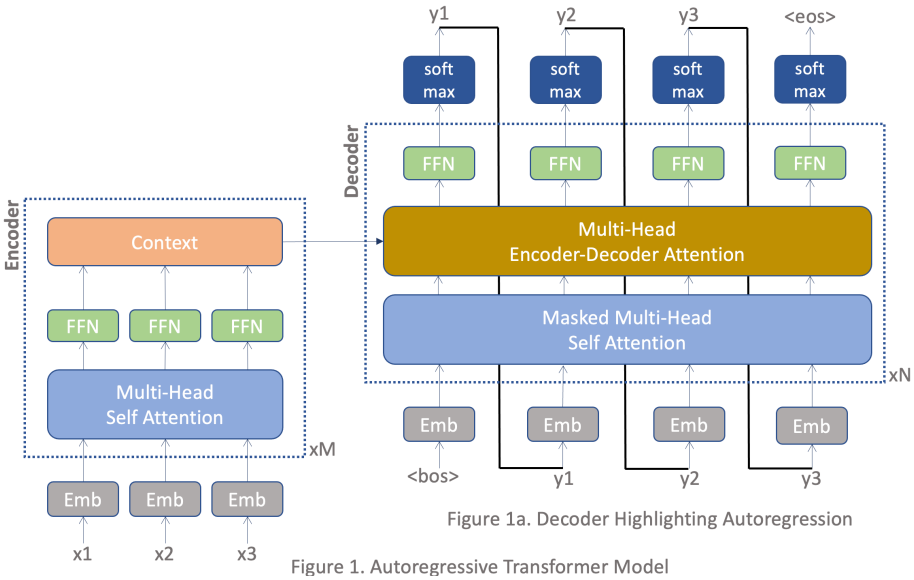

Figure 1a. Decoder Highlighting Autoregression

Figure 1. Autoregressive Transformer Model

As shown in Figure 1, the encoder constitutes a multi-head self-attention module with a full feed forward network that constitutes the "context". This context is directly fed into decoder's attention module (encoder-decoder attention). Decoder's multi-head self-attention is masked to ignore the tokens on the right of the current time step.

During training the decoder's input is known in advance, therefore the masked self-attention layer makes the training process to ignore tokens that cannot be calculated in that specific time step. However, during inference, each token is calculated one at a time as shown in Figure 1a.

The Non-autoregressive model's architecture differs primarily on the decoder side as seen in Figure 2[2].

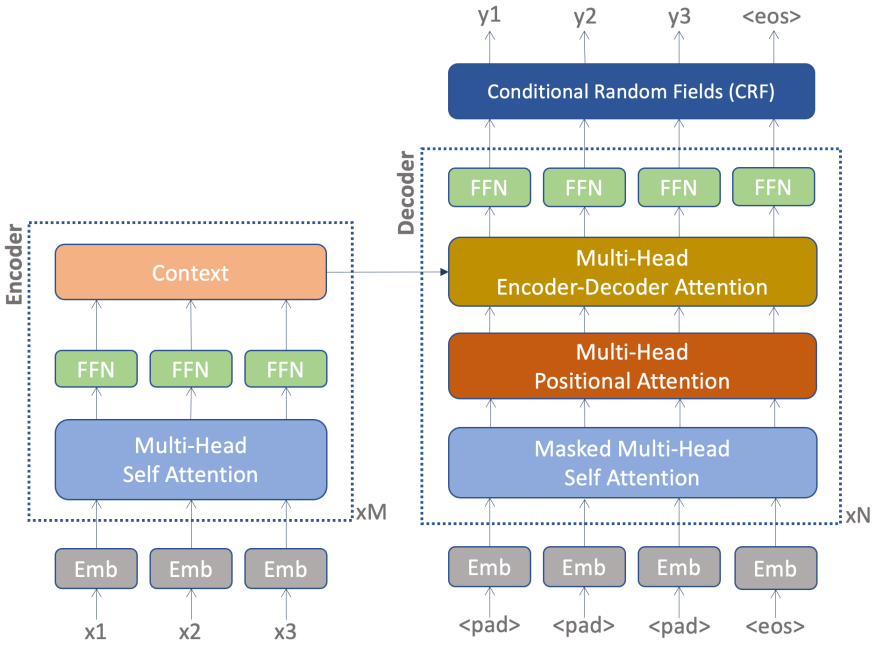

Figure 2. Non-autoregressive Transformer Model with Conditional Random Fields

A new multi-head positional attention layer is introduced in the decoder. The input to the decoder's self-attention layer is simplified by using a sequence of <pad> tokens followed by the <eos> token. The size of the decoder input is predicted based on the size of the input provided to the encoder.

The attention across all modules is calculated based on the following as presented in [3]:

$$Attention(Q, K, V) = softmax\left(\frac{Q.K^T}{\sqrt{d_{model}}}\right).V \qquad (4)$$

Where d_{model} represents the dimensions of the hidden representations.

Following are the values of Q, K, V in each of the attention layers:

- self-attention: Q = K = V = the input x
- encoder-decoder attention: Q = hidden representation of the previous layer; K = V = context vectors from the encoder
- positional attention: Q = K = positional embedding; V = hidden representation of the previous layer

Position-wise Feed-Forward Network (FFN) is applied after multi-head attentions in both encoder and decoder. It is defined as:

$$FFN(x) = max(0, x\,W_1 + b_1)\,W_2 + b_2 \qquad (5)$$

Conditional Random Fields (CRF) is used as a structured inference module. Given input sequence $x = (x_1, x_2, \ldots, x_n)$, and corresponding label sequence, $y = (y_1, y_2, \ldots, y_n)$, the likelihood of y given x is defined as:

$$\mathrm{p}(y|x) = \frac{1}{Z(x)} \exp[\Sigma_{i=1}^{\mathrm{n}} s(y_i, x, i) + \Sigma_{i=2}^{\mathrm{n}} t(y_{i-1}, y_i, x, i)] \qquad (6)$$

where $Z(x)$ is the normalizing factor, $s(y_i, x, i)$ is the label score of y_i at position i and $t(y_{i-1}, y_i, x, i)$ is the transition score from y_{i-1} to y_i.

Following optimizing techniques [2] are used to avoid the computational complexity in the CRF module:

- Low-rank approximation for transition matrix
- Beam approximation for CRF

The CRF module used in the training process uses a negative log-likelihood loss:

$$L_{CRF} = -\log P(y\,|x) \qquad (7)$$

3. Training & Inference

The baseline autoregressive model is built from scratch adopting the transformer model as implemented in the pytorch library. A combination of classes such as transformer, encoder, decoder, encoder layer, decoder layers were customized and extended. spaCy is used as the tokenizer to process the IWSLT text to use "DE" (German) as source and "EN" (English) as the target languages.

We ran the baseline autoregressive model in GPU.LAND[5] environment using Tesla v100 GPU with 16GB memory. Following hyper-parameters setup was used:
number of epochs: 1024; learning rate: 3e-4; batch size: 32; embedding size: 512; number of heads in the attention layer: 8; number of encoder layers: 6; number of decoder layers: 6; activation function: reLu; dropout: 10% (0.1); optimizer: Adam; optimizer betas: (0.9, 0.98)

Running 1024 epochs on the above-mentioned GPU instance took around 20 hours. We recorded a BLEU score of 15.67 after completing the training process. The model was saved as a checkpoint that was later used for the inference yielding a BLEU score results ranging up to 19.

We leveraged Facebook's "fairseq" library implementation to manifest the CRF based non-autoregressive (NAR) transformer model. The CRF and NAR code implementation gave a jump start. The library was modified to incorporate the changes in the decoder architecture[2]. It was however a non-trivial task to make the entire library functional. We used IWSLT dataset with "DE" (German) as source and "EN" (English) as the target languages. The same GPU environment was used on a python 3.6 versioned virtual setup. Following hyper-parameters were used for consistency:
number of epochs: 155; learning rate: 0.0005; optimizer: Adam; optimizer betas: (0.9, 0.98); number of heads in the attention layer: 8; number of encoder layers: 6; number of decoder layers: 6; dropout: 0.3; CRF low rank: 32; CRF beam-approx.: 64

Running with the above config yielded a maximum BLEU score of 9.26 during the training period.

4. Analysis

Table I summarizes the results of the base Auto Regressive (AR) and Non-Auto Regressive (NAR) models. The code, data, and relevant scripts with outputs are available at https://github.com/jroshanucb/deep_learning

TABLE I
MODEL PERFORMANCE

Epochs	Transformer Models	BLEU Score training/validation	Inference BLEU Score
1024	AR Transformer (pytorch based)	15.67	16.07
155	NAR Transformer (fairseq based)	9.26	8.79
50	AR Transformer (fairseq based)	35.23	34.71

The fairseq based AR model results are added in the above table to highlight that the baseline model can be further enhanced exceeding the BLEU scores from the original Vaswani paper[3]. Following are some examples of the translations by both the models. A long list of these translations is available in the github.

TABLE II
TRANSLATIONS

Transformer Models	Text
AR Transformer (pytorch based)	**Source:** ein mann rührt in einem topf in seiner küche . **Predicted:** a man is stirring a pot in the kitchen . **Actual:** a man stirs a pot in his kitchen.
	Source: ein mann in einem roten hemd betritt ein etablissement . **Predicted:** a man in a red inside a small glass . **Actual:** a man in a red shirt enters an establishment.
NAR Transformer (fairseq based)	**Source:** und weil uns nichts wichtiger ist als unser überleben , ist die erste haltestelle für all die informationen ein teil unseres temporallappens , die amygdala **Predicted:** and because nothing is more important to us than survival , the first stop of all of that data is an ancient sliver of the temporal lobe called the amygdala . **Actual:** and because nothing is more important to us than our survival, the first stop for all the information is a part of our temporal lobe, the amygdala

5. Challenges

Besides getting to understand how a transformer is implemented from scratch, we faced quite a few challenges in this project's implementation. Following 3 are noteworthy.

1. We implemented multiple versions of baseline transformer model including writing the attention layer from scratch. However, we finalized leveraging pytorch's implementation and built our code around it
2. Running "fairseq" was not a trivial task. The configuration of the environment, prepping the data, and the knowledge of the execution efficiencies implemented were required. A lot of time spent to understand Cythonize, pyx files and their corresponding .cpp files to successfully run the models
3. As we have to train the models from scratch and be able to change the architecture of a decoder, setting up an infrastructure was huge challenge. GCP, AWS, & Azure environments were explored. We adopted GPU.LAND[5] service that provided the needed flexibility

6. Conclusion

Non-autoregressive models provide latency efficiency gain over the autoregressive models. Due to time-constraints we did not completely get to the scale of the BLEU scores as achieved in the paper [2]. Future MIDS students can further build on this project to close the translation accuracy scores. Hybrid approaches such as autoregressive Teacher combined with non-autoregressive Student based model[6] can be an enhancement as well.

References:

[1] A Study of Non-autoregressive Model for Sequence Generation
Yi Ren, Jinglin Liu,Xu Tan, Zhou Zhao, Sheng Zhao and Tie-Yan Liu

[2] Fast Structured Decoding for Sequence Models
Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin and Zhi-Hong Deng

[3] Attention Is All You Need
Ashish Vasvani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

[4] Fairseq github code: https://github.com/pytorch/fairseq/tree/master/fairseq/models/nat
Baseline non-autoregressive code implementation is in the above fairseq code repository

[5] GPU.LAND – A paid GPU service provider
https://gpu.land/overview

[6] Hint Based Training For Non-Autoregressive Translation
Zhuohan Li, Di He, Fei Tian, Tao Qin, Liwei Wang, Tie-Yan Liu