제 3회 AI X Bookathon 대회 예선 답안 제출

팀명 : Altist
팀 대표 : 이지현
팀원 : 이승윤, 장민석, 성민경

〈 기본 문제 1번 〉

Neural Network를 학습하는 일련의 과정에 대해, 다음 키워드 일부 또는 전부를 활용해 간결히 서술해주세요. (keyword : loss, optimizer, gradient descent, back propagation, learning rate, batch)

NN의 학습은 크게 순전파와 역전파(back propagation)으로 나뉩니다. 먼저 batch size에 따라 분할된 학습 데이터의 subset을 네트워크에 통과시키는 과정인 순전파를 수행합니다. 순전파를 통해 최종적으로 출력되는 예측값(prediction)과 실제값(label)의 비교를 하며, 이 과정에서 사전에 정의된 loss function을 이용해 loss를 계산합니다.

이렇게 loss function에 의해 산출되는 loss를 minimize하는 방향으로 optimizer를 통해 back propagation을 실행합니다. 다시 말해, 학습의 목적은 loss function의 global minimum을 찾는 방향으로 네트워크의 parameter를 update 하는 것입니다. 이때 optimizer를 통해 계산된 gradient에 따라 parameter를 update 할 알고리즘을 적용하고, learning rate를 설정해 한 번의 step마다 parameter를 update 시킬 정도를 결정합니다. 이를 바탕으로 parameter들이 update되는 일련의 과정이 하나의 step입니다. 이 step이 모든 batch에 대해서 순차적으로 이루어져 네트워크가 training data에 포함되어 있지 않은 임의의 input에 대해서도 적은 loss를 갖게 만드는 과정이 학습이라 할 수 있습니다.

[Reference]

- https://www.semanticscholar.org/paper/Learning-representations-by-back-propagating-errors-Rumelhart-Hinton/052b1d8ce63b07fec3de9dbb583772d860b7c769
- 〈문헌〉 Deep Learning (Adaptive Computation and Machine Learning) Goodfellow Ian, Bengio Yoshua, Courville Aaron MIT Press

〈 기본 문제 2번 〉

작년에 공개된 GPT-3는 우리가 이번에 사용할 GPT-2와 무엇이, 어떻게, 얼마나 다를까요?

GPT-3와 GPT-2의 차이점은 parameter scale, attention pattern, few-shot task의 세분화입니다.
GPT-3는 약 1750억개의 parameter로 GPT-2에 비해 상당히 큰 규모입니다. 이와 더불어 word embeddings size와 context window size도 각각 12888, 2048개로 증가했습니다. 즉, GPT-3는 embedding size의 증가로 더 풍부한 표현력을 갖출 수 있으며 window size의 증가로 앞뒤 문맥의 영향을 더 잘 반영할 수 있습니다. 데이터에 있어서도 Common Crawl와 다국어 데이터를 사용하는 등의 더 많은 양을 학습했습니다.

다음으로 attention pattern에 변화를 주었습니다. 기존 transformer의 attention pattern을 sparse transformer에서와 유사한 dense와 locally banded sparse attention pattern으로 대체했습니다. Attention 연산에 필요하지 않은 query와 key의 pair 수를 제한함으로써 문장이 길어질수록 커지는 cost와 dependency 문제를 해소했습니다.

마지막으로 few-shot task에 대한 성능 향상 부분입니다. 해당 부분은 구조적인 차이가 아닌, 논문에서의 실험 방법의 차이로 GPT-2는 zero-shot task를 대상으로 실험을 진행했지만, GPT-3의 경우는 few-shot, one-shot, zero-shot, fine tuning으로 총 4가지로 세분화된 task에 대해 성능을 평가했습니다. zero-shot task에 대해 성능이 좋지 못하여 결국 fine-tuning이 필요했던 GPT-2와 다르게 GPT-3는 더 향상된 성능을 보여주었습니다. 종합하여, 이러한 차이에 기인해 GPT-3는 다양한 task에서 기존 GPT-2를 능가하는 성능을 보여주었습니다.

[Reference]

- https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- https://arxiv.org/abs/2005.14165
- https://openai.com/blog/sparse-transformer/

〈 기본 문제 3번 〉

여러분이 무사히 본 대회에 진출했다고 가정해봅시다. 직접 GPU에서 GPT-2를 finetuning 시키려고 했는데, Out of Memory라는 에러가 떳네요? 예상 가능한 원인들과 그에 따른 해결 방안을 적어주세요.

Out of Memory, 이른바 OOM error는 일반적으로 GPU 장치 내부에 존재하는 GPU RAM(GRAM)의 용량이 초과할 때 발생합니다. 따라서 근본적인 해결 방법은 용량의 초과를 막는 것입니다. 이에 예상되는 원인으로는 두 가지가 있습니다. 모델은 GRAM 용량 안에 들어가지만 학습 과정에서 모델 내부에서 계산되는 data의 용량이 너무 큰 경우와 모델 자체의 용량이 큰 경우입니다. 하지만 GPT-2라면 후자보다는 전자의 원인이 주요할 것입니다.

해결방안은 크게 세 가지 정도가 존재합니다. 우선 가장 기본적인 방법으로는, 사전에 설정하는 batch size를 줄여 내부에서의 연산량을 줄임으로써 해결할 수 있습니다. 또 한가지는 n_positions(maximum sequence length)를 작게 하는 것입니다. 모델에 input으로 사용되는 최대 토큰의 길이를 적절히 제한함으로써 GPU의 사용량을 줄일 수 있습니다.

두 번째로 GPU에 parallelism을 적용하는 것입니다. 여러 개의 GPU 장치 또는 코어에 데이터와 모델을 분산시켜 학습을 하게 되면 각 GRAM마다 모델의 parameter와 학습 데이터가 차지하는 용량이 줄어들어 OOM의 해결과 함께 더 빠른 학습이 가능합니다. 실제로 GPT-3 논문에서는 큰 용량의 모델을 훈련시키기 위해 layer 전반에 걸친 model parallelism을 시행하였습니다. 이러한 parallelism을 위해 pytorch의 torch.nn.DataParallel 나 pytorch-lightening의 Distributed DataParallel 등의 메소드를 사용할 수 있습니다. 추가적으로, 모델을 GPU 연산이 필요한지 여부에 따라 분리하여 CPU와 GPU에 분산하여 올리거나 tensor pipeline에서 parallelism을 적용하는 방법도 있습니다.

마지막으로 mixed precision을 사용하는 것입니다. 일반적으로 딥러닝 모델을 학습 시에는 FP32, 32bit를

활용해 parameter들을 표현하는 방식이 사용됩니다. 이는 parameter의 수가 많아질수록 메모리를 많이 차지하기 때문에 OOM을 유발할 수 있습니다. 대안으로, 16bit를 사용하는 방법이 있으나 이는 정밀도 부분에서 좋지 않아 학습에 있어서 성능 저하를 유발합니다. 이러한 점을 해결하고자 제시된 방법인 mixed preicision은 FP32 weight는 저장해두고, FP16 copy weight을 활용해 학습합니다. 이 과정에서 얻은 gradient를 이용해 FP32 weight를 update하는 방식으로 메모리의 부하를 낮춥니다. 실제로 pytorch에서는 torch.cuda.amp라는 기능으로 구현되어있어 규모가 큰 모델 학습 시에 자주 사용합니다.

[Reference]

- https://developer.nvidia.com/blog/mixed-precision-training-deep-neural-networks/
- https://lilianweng.github.io/lil-log/2021/09/24/train-large-neural-networks.html#model-parallelism

〈 기본 문제 4번 〉

이번 주제는 수필입니다. 그런데, 수필말고 다른 데이터(소설, 시, 신문기사)를 학습한 데이터에 추가하는 것은 모델 생성 결과에 어떤 영향을 미칠까요? 예상되는 장점과 단점을 정리하고, 확인할 수 있는 가설의 경우 확인 방법(실험)을 제시해주세요.

NLP에서는 일반적으로 다양한 유형의 Transfer learning이 사용됩니다. 수필이 아닌 다른 데이터(소설, 시, 신문기사)를 학습 데이터에 추가하는 것은 Same task, different domains, 즉 Domain adaptation에 가까울 것입니다. 이러한 측면에서, 원하는 '수필'이라는 domain이 아니라 '소설, 시, 신문기사'와 같은 다른 domain 데이터는 language modeling의 확장성을 가져올 수 있다는 장점이 있습니다.

언어 모델은 이전 단어들이 주어졌을 때 다음 단어를 예측하도록 함으로써 가장 자연스러운 단어 시퀀스를 찾아냅니다. 언어 모델은 corpus로부터 스스로 확률 값을 결정하기 때문에 일일이 사람이 문법적 정보를 줄 필요가 없는 대신 많은 양의 학습 corpus를 필요로 하는데, 수필 데이터는 그 양이 부족한 편입니다. 따라서 수필 이외의 데이터를 이용하면, 더 많은 단어와 문장 구조를 학습하면서 표현공간을 넓히는 동시에 언어 모델의 일반화 성능을 높일 수 있습니다.

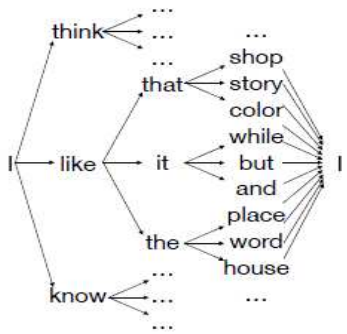
하지만 단점으로는 부적절한 학습을 통해 수필이라는 목적에 부합하지 않는 문장들을 생성할 가능성이 커집니다. 학습 데이터에 포함된 타 문학 작품들의 문체가 지나치게 자주 등장하거나 신문기사와 같이 딱딱한 문체를 생성할 수 있습니다. 특히 신문기사는 사회의 많은 정보가 실려있어 원하지 않는 정보에 대한 전달이 결과물에서 나타날 우려가 큽니다.

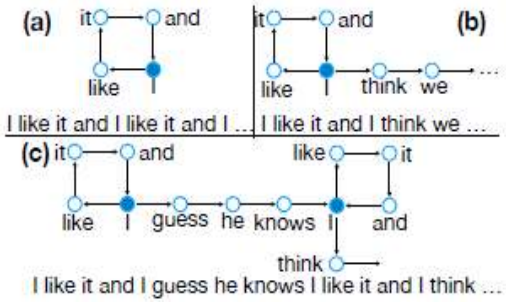
타 데이터의 활용이 문체에 있어서 수필과 거리가 멀다는 가정하에서 이를 확인할 수 있는 실험 방법을 다음과 같이 제시합니다. 학습하는 데이터의 양에 있어서 수필과 수필 이외의 데이터들의 비율을 조정하면서 학습합니다. 예를 들면, '수필 : 소설, 시, 신문기사' 의 비율을 8:2부터 시작하여 수필의 비율을 점진적으로 낮춰가며 모델이 생성하는 결과를 정성적으로 분석하여 보면 이를 확인할 수 있습니다. 수필의 비중이 줄어들수록 더욱 다양한 문체들이 나타날 것이며 수필 특유의 독백과 단어들의 사용은 줄어들 것입니다.

〈 심화 문제 〉

GPT를 사용하다보면 같은 단어 및 문장을 반복 생성하는 경우가 있는데, 이를 repetition problem 이라 합니다. 왜 이런 반복이 일어나는 걸까요? 또한, 반복 문제를 완화하는 방법에는 어떤게 있을까요? 근거를 들어 논리적으로 설명해주세요.

Repetition problem은 GPT뿐만 아니라 언어 생성모델에서 자주 등장하는 문제입니다. 이 문제의 원인은 유사한 문장을 가진 학습 데이터의 중복 또는 학습 중 토큰 간에 closed loop에 해당하는 패턴이 생겨나는 것입니다. 여러 다른 토큰들이 높은 확률로 같은 토큰의 출현을 예측함에 따라, 생성된 closed loop 안에 존재하는 토큰 중 하나가 input으로 주어지면 반복이 지속해서 발생하게 됩니다.





해당 문제를 완화하기 위해서는 크게 decoding 방법의 변화와 Average Repetition Probability(ARP)를 활용한 rebalanced encoding을 사용할 수 있습니다.

첫 번째로 Top-p sampling을 이용할 수 있습니다. 자연어 생성에 있어서 사용되는 decoding 방법은 여러 가지가 있는데, 가장 대표적으로 널리 쓰이는 것이 greedy search와 beam search입니다. 언급한 두 가지 방법은 모두 후보 공간이 작아 반복의 문제가 자주 발생합니다. 이를 Top-p sampling을 이용해 해결할 수 있습니다. Top-p는 토큰들의 출현 누적확률분포를 활용해, 누적확률 합이 설정한 p에 이를 때까지의 토큰 중에서 sampling하는 방법입니다. 이를 통해 closed loop의 생성을 완화하는 동시에 다양한 선택지들을 주어 closed loop가 생성되었더라도 이를 탈출할 가능성을 높일 수 있습니다. 부가적으로 반복되는 ngram size의 설정을 통한 제어도 하나의 방안이 될 수 있습니다.

다음으로 ARP를 활용한 rebalanced encoding입니다. ARP란 모델이 학습한 pattern을 마코프 체인으로 나타냈을 때, 토큰이 다시 출력될 확률을 수치적으로 나타낸 것입니다. 과정은 다음과 같습니다. Encoding된 토큰 사이의 ARP를 계산한 transition matrix를 만듭니다. 이 matrix에서 일정 값 이상의 높은 확률을 가지는 토큰 쌍이 발견될 경우, 다시 말해 토큰 쌍에서 첫 번째 토큰이 입력된 경우에 두 번째 토큰이 출력될 확률이 일정 값보다 클 경우, 두 토큰을 하나의 토큰으로 합성합니다. 이렇게 토큰들을 합성하는 과정을 거치면 transition matrix에서 하나의 토큰이 필연적으로 다른 토큰을 출력하게 하는 현상을 줄일 수 있고 repeating token 문제를 해결할 수 있습니다.

[Reference]

- https://arxiv.org/abs/1904.09751
- https://huggingface.co/blog/how-to-generate
- https://arxiv.org/abs/2012.14660