파이토치를 활용한 분산 임베딩 프로젝트

Sungwon Lyu
데이터마이닝 연구실
서울대학교 산업공학과
lyusungwon@dm.snu.ac.kr

Jeeyoung Kim
데이터마이닝 연구실
서울대학교 산업공학과
jeeyoung.kim@dm.snu.ac.kr

Noori Kim
데이터마이닝 연구실
서울대학교 산업공학과
noori@dm.snu.ac.kr

Jihun Yi
데이터사이언스/인공지능 연구실
서울대학교 공과대학 전기정보공학부
t080205@gmail.com

개요

본 논문에서는 텍스트 임베딩의 가속화를 위해 스트리밍 텍스트 분산 임베딩 프레임워크와 RNN 기반 단어 임베딩을 제안하여 이를 통해 실험을 진행하고자 하였다. 하지만 실험을 진행하는 과정에서 안정적인 스트리밍 데이터 소스를 구하는 것에 대한 어려움에 부딪히게 되었다. 이에 따라 연구의 방향을 분산 임베딩의 유효성을 확인하는 것으로 수정하였고 pip loss 를 활용하여 임베딩의 품질을 실시간으로 확인하였다. 본 연구의 의의는 파이토치 프레임워크를 활용하여 간단한 텍스트 데이터 셋에 대해 다양한 환경에서 분산 학습을 진행한 결과를 정리한 것으로 볼 수 있다.

키워드

단어 임베딩, 딥러닝 분산 학습, RNN 단어 임베딩 생성기, pip loss, 파이토치 분산 학습

1 연구 배경

Natural Language Processing(NLP) 분야에서 다루고 있는 QA(Question Answering), NER(Named Entity Recognition), SRL(Semantic Role Labeling) 등의 다양한 문제들을 풀기 위해 선행되어야 하는 부분은 각 단어를 벡터로 표현하는 것이다. 단어를 다차원 공간 상에서 벡터로 표현하여 나타내는 것을 Word Embedding 이라고 하며 이는 과거부터 활발히 진행된 연구이다. 대표적인 Word Embedding 방법에는 Skip-gram[1], Cbow[2], Glove[3], FastText[4]등이 있다.

하지만 기존의 Word embedding 방법들은 몇 가지 문제점을 가지고 있다. 우선, 변화가 반영되지 않는 고정된 문서 데이터를 이용하여 학습을 진행하기 때문에, 변화하는 언어의 흐름을 반영하는데 한계가 있다. 두 번째로, 학습 데이터의 corpus 에 없는 단어는 embedding 할 수 없는 Out of Vocabulary(OOV)의 문제점이 존재한다. 마지막으로, Word embedding 을 위해 stemming, lemmatization 등의 텍스트 데이터에 대한 전처리가 필요하며, 이는 시간과 자원이 매우 많이 소요되는 작업이다.

향후 장에서 소개될 내용은 다음과 같다. 2 장은 관련 연구, 3 장은 방법 및 실험 환경 설정, 4 장은 결과 해석 그리고 5 장은 결론 도출로 구성되어 있다.

2 관련 연구

2.1 단어 임베딩

신경망을 통한 단어 임베딩 알고리즘은 현재 광범위하게 사용되고 있다. 중심단어로 부터 주위단어를 예측하는 Skip-gram[1]은 효율적인 연산을 위한 부정 표본 추출 기법과 결부되어[2] 을 사용하여 현재 가장 보편적으로 사용되고 있다(Skip-Gram Negative Sampling: SGNS). 이후 SGNS 에서 단어 발생빈도에 대한 정보를 반영하기 어렵다는 점을 보완한 GloVe[3]가 제안되었으며 최근에는 텍스트 분류를 위한 목적으로 N-gram 단위의 작은 단위를 임베딩 한 후에 이를 조합하여 사용하는 FastText[4]가 사용되고 있다.

이러한 알고리즘을 통한 연속적 단어 표현이 좋은 성능을 보이는 것에 대한 여러 연구가 진행되었다. 위와 같은 기법들을 통하여 임베딩된 단어들은 비슷하게 사용되는 단어들 끼리 모여있는 경향을 보일 뿐만 아니라 벡터화된 단어들 간의 차이에서 의미론적, 구문론적 규칙을 찾아내는 것을 보여주었다[5, 6].

[7]에서는 인공 신경망으로 학습된 단어의 임베딩이 궁극적으로는 전통적으로 사용되었던 단어의 발생 통계를 반영한다는 것을 밝혔고 [8]에서 인공 신경망 계열의 단어 임베딩 알고리즘들의 우수한 성과는 몇 가지의 구조 선택과 하이퍼파라미터 최적화 덕분이라는 것을 밝혀내었다.

2.2 분산 딥러닝 학습

기존의 데이터 분산 처리 프레임워크들은 데이터 마이닝이나 기계 학습에 초점을 맞추어서 이루어지고 있었다. 많은 양의 데이터와 연산에 대하여 하나의 좋은 장비 대신 군집의 장비를 활용하여 분산 처리를 하기 위한 프레임워크들이 등장하여 보편적으로 사용되고 있다[9, 10, 11]. 하지만 이러한 분산 프레임워크들은 딥러닝에서 필수적인 역전파의 구현이 제한되어 딥러닝에서는 전처리 역할에 그쳤다. 딥러닝이 발전하기 시작하면서 딥러닝을 위한 분산 방법들이 연구되기 시작하였다.

[14]는 딥러닝을 위한 병렬로서 모델 분산과 데이터 분산을 제안하였다. 모델 분산은 모델의 크기가 너무 클 경우 여러 장비에 모델을 나누어 담고 이를 학습하는 것이다. 반대로 데이터 분산은 한 모델에 대하여 많은 데이터를 학습해야 할 경우 여러 장비에 데이터를 나누어 담아서 학습하는 방식이다. 이 두 가지 방법을 혼용하여 사용할 수 있다.

데이터 분산을 위해서 여러 장비에 한 모델을 복사해서 각각 학습하게 되는데 이 복사본들의 동기화를 하는 방식이 딥러닝 분산 학습의 중요한 주제이다. [14]에서는 두 가지 동기화 방식으로 동기적 방식인 Downpour SGD 와 비동기적 방식인 L-BFGS 을 제안한다. Downpour 방식은 각 장비가 학습을 한 후 그래디언트를 파라미터 서버에 보내어 파라미터 서버를 비동기적으로 관리한다. 반면 L-BFGS 방식은 Coordinator 가 메시지를 통하여 파라미터 서버와 모델 복사본들을 관리하여 동시에 학습한다.

비동기적 방식의 장점은 시간 당 처리하는 데이터의 양이 많다는 점이다. 각 장비가 학습하는 시간의 편차가 있을 수 있는데 각 장비가 빨리 학습하는 대로 파라미터를 갱신하여 다음 학습을 할 수 있기 때문이다. 다만 이 때문에 파라미터 서버와 장비의 통신이 많아질 수 있다. 기존에는 이러한 통신을 할 때 마다 파라미터 서버 내용을 안전하게 보존하기 위하여 다른 통신들의 접근을 제한하는 잠금이 이루어 졌는데 [15]에서는 한 장비가 통신을 할 때 잠금을 하지 않더라도 문제가 생길 확률이 낮고 오히려 갱신이 효율적으로 이루어 질 수 있다는 점을 보여주었다. 이러한 이유로 과거의 딥러닝 분산에는 비동기적 방식이 주를 이루었다.

모든 장비의 그래디언트 계산이 끝나야만 파라미터를 업데이트 할 수 있어서 가장 느린 장비의 학습 속도에 맞춰진다는 동기적 방식의 특성 때문에 딥러닝의 분산 학습에서 동기적 방식보다 비동기적 방식이 선호되었다. 하지만 점차 목적 함수의 수렴 측면에서는 비동기적 방식보다 동기적 방식이 더 효율적이기 때문에 동기적 방식이 데이터 효율이 더 좋다는 관찰이 보고되면서 동기적 방식이 선호되기 시작하였다. 이와 더불어 위에서와 같이 하나의 장비가 전체의 학습 속도를 늦추는 현상을 방지하기 위하여 [16] 에서는 여러 장비 중 빠르게 학습되는 k 개 장비의 결과만 활용하여 학습하는 방법을 제안하였다.

동기 방식이 비동기 방식에 비하여 효율적인 이유는 각 장비가 그래디언트를 구하는 동안 다른 장비들이 파라미터를 갱신하기 때문에 그래디언트를 구한 파라미터와 업데이트 하려 파라미터간의 괴리가 생기기 때문이다. [17]에서는 이러한 차이를 최소화 하여 동기 방식과 비동기 방식의 중간 형태인 제한된 동기 방식을 제안하였다.

동기화 방식과는 별개로 분산 학습 구조는 분산 학습의 결과에 중요한 영향을 미친다. 분산 학습의 구조는 크게 All-reduce/All-gather 방식과 파라미터 서버 방식이 있다. All-reduce/All-gather 방식은 서버를 따로 두지 않고 각 워커들이 다른 데이터들에 대하여 각각 학습한 후 워커들 간의 집단 통신(All-reduce/All-gather)을 통하여 동기화 하는 방식이다. 실제로 All-reduce/All-gather 와 같은 오퍼레이션은 사용하지 않고 Unidirectional Ring 과 같은 방식을 통하여 동기화한다. 파라미터 서버 방식은 말 그대로 워커의 외부에 파라미터를 담당하는 서버를 두고 워커들이 이 서버들과 통신하는 방식을 의미한다. [18]에서는 이러한 방식으로 학습할 때 업데이트 하는 그래디언트의 밀도가 희박하다는 점에서 착안하여 그래디언트를 계층화(Quantization)하고 인덱스만을 저장하여 서로에게 전파하는 방식을 제안하였다.

[19] 에서는 이러한 All-reduce/All-gather 구조와 파라미터 서버의 구조의 성능을 측정해 보았는데 특이한 발견을 제보하였다. 파라미터 업데이트의 분포가 밀집한 모델의 경우에는 All-reduce/All-gather 구조가 효과적이었고 분포가 희박한 경우에는 파라미터 서버 구조가 효과적이었다. 그리하여 [19]에서는 파라미터의 분포에 따라서 구조를 선택하는 혼합 구조를 제안하였다.

딥러닝의 분산 학습을 위해서는 위와 같은 고려 사항들이 있기 때문에 모델의 구조와 데이터를 보고 적절하게 선택하여야 한다.

2.3 스트리밍 딥러닝 학습

기존의 데이터 처리 프레임워크에서는 지속적으로 쌓이는 데이터들을 처리하기 위한 스트림 처리 프레임워크[12, 13]들이 등장하여 산업에서 활용되고 있다. 이러한 스트림 처리 프레임 워크들은 데이터의 생성 시간과 도달 시간 사이의 간극을 메우는 것이 중요한 과제였다. 하지만 딥러닝에서는 일반적으로 데이터가 모델로 학습하는 과정에서 데이터의 생성 시간이 중요하지 않다. 또한 데이터의 학습과 추론이 따로 이루어 진다는 점도 딥러닝의 스트림 처리에 대한 연구가 희박한 이유이다.

딥러닝에서 스트림 데이터의 처리가 필요하다면 변화하는 데이터를 빠르게 반영해야하기 때문이다. 이와 같은 환경은 환경과 지속적으로 상호작용하는 강화 학습 환경이 있다. 강화학습은 환경에서 얻은 경험을 통하여 모델을 학습하고 학습한 모델이 환경에 영향을 주어 다른 경험을 얻기 때문에 환경에서 나오는 데이터들을 즉각적으로 반영해야 한다. 이러한 측면에서는 강화학습 환경이 스트림 처리가 필요한 환경이라고 볼 수 있다. 또한 이러한 경험들을 최대한 빠르게 많이 학습해야 하기 때문에 강화학습에서 분산을 활용하려는 접근이 많이 있었다.

[20] 은 강화 학습에 딥러닝 분산 학습을 적용하였다. 파라미터 서버와 행동자, 학습자를 나누어 행동자들이 수집한 데이터를 재생 기억이라는 저장소에 저장한 후 학습자들은 이를 학습하기만 하도록 분업화하였다. 이 군집 환경에서 비동기적 분산 학습을 통하여 강화 학습을 빠르게 학습시키는데 성공하였다. [21]에서는 이러한 분산 학습을 통하여 단순히 학습을 속도를 향상 시킬 뿐만 아니라 환경으로부터 수집하는 데이터의 편향을 극복하고 다양화하기 위하여 활용하였다.
강화 학습 환경에서는 데이터의 경험을 여러 번 재사용하기 위하여 데이터들을 재생 기억이라는 저장소에 넣어두고 이를 무작위 추출하여 학습하는 방식을 취한다. 하지만 점차 효율적인 학습을 위하여 단순 표본 추출이 아닌 중요도 기반 표본 추출이 제안되었다. 데이터의 중요도를 판단하기 위하여 [22]에서는 해당 데이터의 그래디언트의 L2 노름을 활용하기도 하고 [23]에서는 해당 데이터의 손실 값을 활용하기도 한다.
[24]에서는 분산 학습과 데이터의 중요도 기반 추출 방식을 모두 활용하여 시간 대비 최고의 성과를 보여주었다. 여러 행동자들은 지속적으로 활동하며 비동기적으로 데이터를 경험 재생에 넣는다. 반면 하나의 학습자는 경험 재생에 쌓인 데이터를 주기적으로 추출하여 학습한다. 이 때 단순 무작위 추출이 아닌 우선순위를 기반하여 추출하여 학습하기 때문에 효율적인 학습이 가능하다. 이러한 구조가 효율적인 이유는 학습자는 하나의 GPU 가 필요하지만 행동자들은 CPU 만 필요로 하기 때문에 여러 행동자가 활동하는 것이 가능하기 때문이다. 본 논문의 방법론은 이 논문으로부터 많은 영감을 받았다.

3 제안 방법

본 프로젝트에서는 기존의 Word embedding 이 갖고 있는 문제를 해결하고자, 새로운 word embedding system 을 제안하려고 했다. 해당 모델은 기존의 연구와 두 가지 측면에서 차별점을 갖는다.
우선, 기존에 갱신이 되지 않는 문서 데이터를 이용한 것과 달리 crawler 를 이용하여 계속해서 갱신되는 데이터를 학습에 이용한다. 따라서 언어의 변화를 반영하는 Word embedding 이 가능해진다.

3.1 스트리밍 텍스트 분산 임베딩 프레임워크

다양한 출처의 인터넷 문서를 효율적으로 임베딩하기 위하여 데이터 분산 딥러닝 학습을 제안한다. 임베딩은 모델 중에서도 파라미터의 갱신의 분포가 매우 희박하기 때문에 All-reduce/All-gather 구조보다는 외부에 파라미터의 정보를 가지고 있는 서버를 둔 파라미터 서버 구조가 적합할 것이다. 하지만, 본 논문 실험을 위해 사용한 pytorch 에서는 All-reduce 방식을 제공하기 때문에 MPI 방식을 사용했으며, parameter server 를 사용할 경우 성능 향상이 있을 것이라 기대한다. 제안하는 스트리밍 텍스트 분산 임베딩 방법은 아래와 같다.
CPU 는 각 노드에 할당된 인터넷의 출처로부터 크롤링을 한다. 크롤링한 데이터는 디스크에 저장하지 않고 즉시 전처리하여 또 다른 프로세스를 위한 큐에 넣는다. 다른 프로세스는 크롤링을 하는 프로세스로부터 받은 데이터를 가지고 주어진 모델에 대하여 그래디언트를 계산하고 한 배치를 학습할 때 마다 다른 노드들의 그래디언트를 받아와서 평균을 내고 그 그래디언트를 각 노드들이 업데이트 한다.
동기화 방식은 각 노드들이 할당받은 크롤링 출처에 따라 달라질 수 있다. 만약 각 출처들의 한번에 긁어오는 데이터의 양들이 비슷하다면 데이터의 효율을 최대화 하기 위하여 동기적 방식을 사용할 수 있다. 반면 각 출처 별로 학습 속도의 편차가 커서 전체 학습 속도를 둔화할 것 같다면 큐를 이용하여 비동기적 방식을 사용할 수 있을 것이다.
각 노드들은 크롤링한 데이터들을 재생 기억과 같은 데이터 저장소에 보관해 놓고 학습하기 때문에 강화 학습에서 사용한 방법들과 같은 유사한 기법을 사용할 수 있다. 본 논문의 실험에서는 이 데이터들의 우선 순위를 따로 정하지 않지만 후속 연구에서 모델을 효율적으로 학습하기 위한 우선 순위를 설정할 수 있을 것이다.

3.2 스트리밍 텍스트를 위한 임베딩 방법

3.2.1 Skip-gram

Skip-gram 은 비슷한 위치에 등장하는 단어의 의미는 유사함을 이용하여 학습을 한다. 중심 단어가 주어졌을 때, 주변 단어가 나올 확률은 최대화 되고, 주변 단어가 아닌 단어들이 나올 확률은 최소화 되는 방향으로 학습을 한다. 아래와 같은 목적 함수가 최소화 되도록 학습한다.

$$\mathcal{J} = \log P(w_{c-m}, \cdots, w_{c-1}, w_{c+1}, \cdots, w_{c+m} | w_c) \quad (1)$$

$$\mathcal{J} = -\sum_{j=0, j\neq m}^{2m} \log P(u_{c-m+j} | v_c) \quad (2)$$

Skip gram 모델을 학습시킬 때는 모델의 효율적인 학습을 위해 negative sampling 방법을 사용한다. Negative sampling 을 사용하지 않고 위와 같은 모델을 학습시키면 단어의 수에 따라서 계산 량이 매우 커지게 되며, 더불어 학습도 잘 되지 않는다. 따라서 window 에 포함되지 않은 단어에 대해서는, 말뭉치의 단어 빈도수를 고려하여 sampling 을 하고, 이러한 sample 들만 loss function 계산에 이용하여 효율적으로 loss 를 계산할 수 있다. Negative sampling 을 사용한 목적 함수는 아래와 같이 정의된다.

$$\mathcal{J} = -\log \sigma\left(u_{c-m+j}^T v_c\right) - \sum_{k=1}^{K} log\sigma(-\tilde{u}_k^T v_c) \quad (3)$$

[7]에서 밝힌 것처럼 위의 방법으로 학습된 임베딩은 문서 내 단어의 반생 통계를 반영하게 된다.

3.2.2 RNN word embedding generator

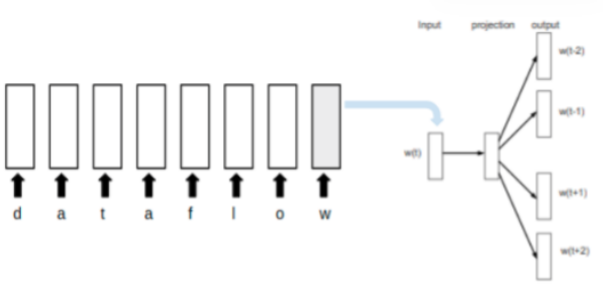

그림 1 – 제안하는 모델의 전체 모식도

빈번히 쓰이는 RNN 모델로는 GRU(Gated Recurrent Unit), LSTM(Long Short Term Memory) 가 있으며, 본 논문에서는 LSTM 을 사용했다.

위 그림 1 과 같이 dataflow 라는 단어를 LSTM 에 입력시키는데, d, a, t, a, f, l, o, w 를 embedding 하여 입력한다. 위의 예시와 같은 경우, sequence length 는 8 이며 마지막 hidden state 인 h_T를 단어 임베딩의 결과로 얻는다. 마지막 hidden vector 에는 단어의 형태학적 특징이 encoding 되었음을 기대한다.

$$h_t = \sigma(W_{ce}^{hh} h_{t-1} + W_{ce}^{hw} x_t) \qquad (4)$$

$x_t \in \mathcal{R}^d$, $W_{ce}^{hx} \in \mathcal{R}^{D_h \times d}$, $W_{ce}^{hh} \in \mathcal{R}^{D_h \times D_h}$, $h_{t-1} \in \mathcal{R}^{D_h}$ 이다. W_{ce}^{hx}와 W_{ce}^{hh}가 학습 가능한 parameter 이다.

중심 단어, 주변 단어 및 negative sampling 각 단어의 분산 표현을 LSTM 을 통해 얻는다. 중심 단어 LSTM 과 주변 단어 및 negative sampling LSTM, 총 2 개의 LSTM 을 각각 학습한다. 중심 단어 LSTM 은 W_{ce}로 표기하였고, 위 수식이 주변 단어 LSTM 에도 똑같이 적용된다.

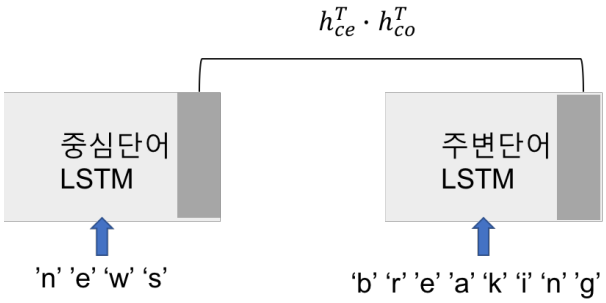

그림 2 - 중심단어 LSTM 및 주변 단어 LSTM

예를 들어, 'breaking news analysis is important for national security' 문장에서 news 가 중심 단어로 설정하고, window size 를 1 로 설정하면, (news, breaking), (news, analysis)와 같은 (중심단어, 주변단어) pair 를 얻는다. 그림 2 와 같이 각각을 LSTM 에 입력하여 계산한다.

LSTM 의 parameter 가 전체 단어 임베딩 정보를 학습할 수 있도록 D_h를 일반적인 LSTM 의 hidden vector 보다는 크게 설정한다.

최종적으로, LSTM 의 output 을 각각 fully-connected layer 를 통과시켜 단어의 임베딩을 얻는다.

목적함수

본 논문에서 제안한 모델을 학습시킬 때의 목적 함수는 기존 연구에서 언급한 Skip-gram 의 것과 유사하다. Skip-gram 에서는 단어를 보고 주변 단어를 예측하게 된다. 중심 단어는 w_c , 주변 단어는 $w_{c-m}, \cdots, w_{c-1}, w_{c+1}, \cdots w_{c+m}$으로 표기한다.

기존의 Skip-gram 에서는 이미 학습이 된 단어 임베딩을 look-up 하여 w를 나타낸 반면에, 본 논문에서 제안하는 모델에서는 RNN 의 결과물인 h_T로 각 단어들을 나타낸다.

최종 목적 함수는 (1) 및 (2)와 유사하지만, w가 h_T 로 바뀌었다.

$$\mathcal{J} = -\log P(h_T^{c-m}, \cdots, h_T^{c-1}, h_T^{c+1}, \cdots, h_T^{c+m} | h_T^c) \qquad (5)$$

h_c^T 는 중심 단어의 LSTM 마지막 hidden vector h_T 이다. 단어 임베딩 생성기 또한 skip-gram 에서 제안한 negative sampling 방법을 이용하여 학습한다.

본 논문에서 제안하는 모델의 학습 결과로, 모델의 RNN 모듈에 해당하는 단어 임베딩 생성기를 얻게 된다. 기존의 모델과는 달리 단어의 구문론적, 의미론적 특성뿐만 아니라 단어 자체의 형태적 특징을 고려하여 임베딩 하였기 때문에 더 정밀한 임베딩 결과를 얻을 것이라 기대한다. 또한, 학습의 결과로 생성기를 얻기 때문에 새로운 단어(OOV)가 왔을 때, 기존에 아무 정보를 담지 못하는 초기화 방법과는 달리 보다 효과적으로 단어를 표현할 수 있다. 더불어, 이러한 접근은 새로운 단어가 생길 때마다 새롭게 전체 단어 임베딩을 학습할 필요가 없기 때문에 시간과 자원을 아낄 수 있다는 장점도 갖는다.

지금까지 언급했던 내용들을 실제로 구현한 후 결과를 도출하는 것을 본래 프로젝트의 목표로 삼았으나 현실적인 문제점들에 부딪히며 목표했던 결과를 얻지 못했다.

첫 번째로, 현실에 안정적인 스트리밍이 가능한 데이터셋이 없다는 것이었다. 우선 실험에 사용할 수 있는 머신의 숫자가 적었으며, 페이스북, 유튜브, 트위터를 crawling 하는 API 들의 안정성이 떨어졌다. 또한 GPU 에서 발생하는 병목 현상보다 crawling 자체를 하는 과정에서 발생하는 병목 현상의 영향이 더 컸다. 이러한 제약 조건들 때문에 당초 설계했던 실험의 목표를 분산 Word embedding 과 모델의 타당성을 확인하는 것으로 변경할 수 밖에 없었다.

두 번째로 제안한 모델에 대한 명확한 평가 기준이 존재하지 않는다는 점이었다. Word embedding 의 성능을 측정하는 기준으로 Word similarity task 와 Word analogy task 가 존재한다. 하지만 성능을 제대로 측정하기 위해서는 특정 분야에 편중되지 않고 모든 단어들을 포함하는 데이터셋으로

학습이 진행되어야 한다. 하지만 이러한 보편성을 만족한다고 판단할 수 있는 위키피디아 데이터셋의 경우 텍스트 원문이 32GB 를 차지하며, 이를 전처리할 경우 용량이 10 배 증가하여 320GB 라는 대용량의 데이터가 되는데 이를 처리하는데에는 현실적인 조건으로 불가능했다. 이에 대한 대안으로 PIP loss[5]라는 평가 기준은 차용했다. 이는 embedding 들 사이의 거리를 측정하는 기준으로 embedding 의 특성 중 단일적인 불변성을 이용한 것이다. PIP loss 의 식은 다음과 같다.

$$\mathrm{PIP}(E) = EE^{T}, \| \mathrm{PIP}(E_1)\text{-}\mathrm{PIP}(E_2) \| = \| E_1E_1^{T}\text{-}E_2E_2^{T} \| = \sqrt{\sum_{i,j}\left(< v_i^{(1)}, v_j^{(1)} > - < v_i^{(2)}, v_j^{(2)} >\right)^2} \quad (1)$$

또한 Skip-gram 의 기준으로 삼을 수 있는 점수에는 SPPMI matrix[6]가 존재한다. SPPMI 의 식은 다음과 같다.

$$\mathrm{PMI}_{ij} = \log\frac{P(v_i,v_j)}{P(v_i)\,p(v_j)} \quad (2)$$

SPPMI matrix 를 통해 구한 PIP loss 를 본 실험의 평가 기준으로 삼았다.

마지막으로 실험을 구현하기 위해 사용한 프레임워크인 Pytorch 가 분산 처리를 지원하는데 있어 불안정하다는 점이었다. 비록 최근에 업데이트 버전인 1.0 버전이 출시되었지만 그럼에도 불구하고 분산처리에 불안정한 모습을 보이는 건 아직 완벽하게 해결되지 않았다.

4. 실험 방법

4.1 전처리 및 모델 parameter

실험에 이용한 dataset 은 6MB 의 Harry Porter 데이터이다. 실험의 Skip-gram 모델과 character-wise LSTM word embedding 두 모델을 이용하여 학습을 진행했다. 두 모델은 총 300 epoch 학습했다.

임베딩 생성기의 평가를 위해, PIP loss 를 이용하여 학습 진행 정도를 평가 했으며, 학습 loss 는 logsigmoid 를 사용했다.

Dataset 전 처리의 경우 다음과 같은 hyper-parameter 를 설정했다. 10 번 negative sampling 을 했고, 등장 빈도수가 높은 단어들을 제거하기 위해 $2 * 10^{-3}$의 threshold 를 사용하여 subsampling 을 했다. 또한 3 번 이하로 드물게 등장하는 단어는 제거 했다.

모델 학습의 parameter 는 다음과 같다. Baseline 실험으로 사용한 skip-gram 의 경우, embedding size 는 200, 50 을 사용했다. learning rate 는 0.0001 로 설정했다.

본 논문에서 제안한 RBSG(RNN Based Skip-gram)의 경우, character embedding size 128, LSTM hidden size 512, learning rate 는 0.0001 로 실험을 했다.

모델 평가 시, Inference 에 사용한 RBSG 의 LSTM 모듈은 중심 단어 LSTM 과 주변 단어 LSTM 두개 중 중심 단어 LSTM 을 사용했다.

두 모델 다 optimizer 로는 Adam 을 이용했다.

4.2 cluster 실험 환경

분산 학습 실험은 총 4 가지 환경에서 진행했다.

Single-node CPU 만을 이용하여, 1 process 처리, single-node single-GPU 환경에서 1 process 처리, single-node, 4-GPUs 환경에서 1 process 처리 및 multi-nodes(4), single-GPU each node 환경에서 4 개의 process 를 처리했다.

학습에 사용한 각 node 의 spec 은, 다음과 같다.

CPU 는 master 의 경우 i5-4570 CPU @ 3.20GHz, 4 cores 이며, slave node 의 경우 i7-4790 CPU @ 3.60GHz, 8cores 를 사용했다. 각 노드의 GPU 는 GeForce GTX 970 을 사용했고, 메모리는 4GB 이다. Single-node, 4-GPUs 환경의 spec 은 CPU 의 경우에는, Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 32 cores 이며 GPU 는 11GB GeForce GTX 1080 을 사용했다.

4 개의 node 에서 실험을 진행할 때는, Ethernet 을 이용해서 실험을 진행했으며, node 의 gradient update 를 위해서 all-reduce 방식을 사용했다. Synchronous 하게 gradient 를 update 하는 경우와, Asynchronous 하게 gradient 를 update 하는 경우를 평가했다.

5 실험 결과 및 해석

다음은 총 4 가지의 실험으로 얻은 결과이다. 미니 배치의 크기와 임베딩의 크기를 각각 상대적으로 작은 경우와 큰 경우를 비교해 보았다.

5.1 실험 1 (작은 미니 배치, 큰 임베딩)

Table 1 실험 1 의 결과 (임베딩 크기: 200, 미니 배치: 1024)

	Average time per epoch (s)	Throughput (data/s)	Best PIP loss
1process 1 GPU	34.10	98,212.7	123.6
1process 4 GPUs	25.37	132,060.5	129.6
Cluster	**394.27**	8,494.3	?

GPU 의 개수가 x4 배 됨에 따라 훈련 속도는 x1.34 배에 그쳤다. 또한 클러스터를 활용한 실험에서는 오히려 훈련 속도가 x0.09 배가 되는 등 속도에 심각한 저하가 발생했다.

아래의 훈련에 따른 loss 의 양상은 훈련이 잘 진행되었음을 보여준다.

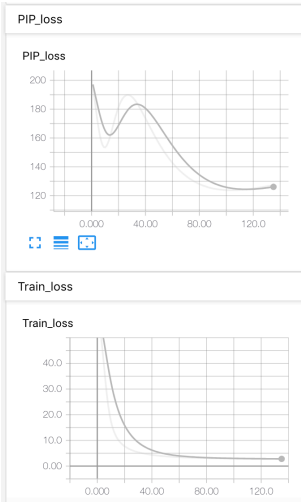

그림 3 실험 1 에서 1 개의 GPU 를 사용했을 때의 PIP loss 와 훈련에 사용한 loss 의 변화

5.2 실험 2 (큰 미니 배치, 큰 임베딩)

Table 2 실험 2 의 결과 (임베딩 크기: 200, 미니 배치: 8192)

	Average time per epoch (s)	Throughput (data/s)	Best PIP loss
1process 1 GPU	28.6	117,099.8	129.3
1process 4 GPUs	24.1	138,964.9	-
Cluster (Sync)	52.79	63,441	193.6
Cluster (Async)	46.5	72,022.6	?

GPU 의 수가 x4 배 됨에 따라 훈련 속도는 x1.19 배로 빨라졌다.
클러스터를 사용한 경우, 훈련 속도는 x0.54 배로 더 느려졌다. 하지만 이는 실험 1 의 x0.09 배와 비교하면 더 나은 수치로, 미니 배치의 크기가 증가함에 따라 통신 오버헤드가 줄어들었기 때문인 것으로 보인다.
또한 클러스터를 활용해 asynchronous 하게 훈련시킬 경우 훈련 속도는 x0.62 배로 synchronous 한 경우보다 더 빨라졌다. 하지만 수렴에는 실패하였다. 이는 아래의 그림에도 드러난다.

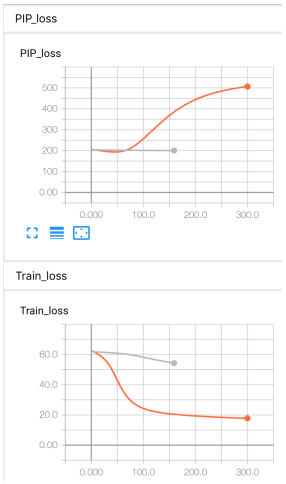

그림 4 실험 2 에서 클러스터를 사용했을 때 PIP loss 와 훈련에 사용한 loss 의 변화. Async 방식(주황색), sync 방식(회색)

5.3 실험 3 (작은 미니 배치, 작은 임베딩)

Table 3 실험 3 의 결과 (임베딩 크기: 50, 미니 배치: 1024)

	Average time per epoch (s)	Throughput (data/s)	Best PIP loss
1process 1 GPU	21.6	155,048.8	14.52
1process 4 GPUs	24.08	139,080.3	15.44
Cluster	93.81	35,700.4	44.21

GPU 의 수가 x4 배 됨에 따라 훈련 속도는 x0.90 배로 더 느려졌다.
클러스터를 사용했을 때에는 x0.23 배로 느려졌다. 이는 실험 1 의 x0.09 배에 비하면 나은 수치인데, 임베딩의 크기가 작아져 통신 오버헤드가 줄어들었기 때문인 것으로 보인다.

5.4 실험 4 (큰 미니 배치, 작은 임베딩)

Table 4 실험 4 의 결과 (임베딩 크기: 50, 미니 배치: 8192)

	Average time per epoch (s)	Throughput (data/s)	Best PIP loss
1process 1 GPU	29.32	114,224.2	15.19
1process 4 GPUs	21.28	157,380.3	-
Cluster	**16.93**	197,817.7	44.12

GPU 의 수가 x4 배 됨에 따라 훈련 속도는 x1.38 배로 빨라졌다.
클러스터를 사용한 경우, 훈련 속도는 x1.73 배로 더 빨라졌다. 이는 총 4 개의 실험 중 클러스터를 이용해 유일하게 훈련이 가속된 경우로, 임베딩의 크기가 작고 배치 크기가 크기

때문에 통신 오버헤드가 상당히 줄어들어 연산 유닛의 개수 증가가 효과를 나타낸 것으로 보인다.

6 결론

6.1 훈련 속도

훈련 속도에 영향을 주는 요인은 연산 유닛의 개수와, 그들간의 통신 오버헤드이다. 더 큰 미니 배치를 사용할수록 같은 수의 데이터를 처리할 때 통신은 덜 자주 일어나고, 더 작은 모델을 사용할수록 통신의 오버헤드는 작아진다. 따라서 더 큰 미니 배치를 사용하고 더 작은 모델을 사용함에 따라 훈련 속도가 증가하는 것을 확인할 수 있었다.

6.2 클러스터의 효용

총 4 가지의 실험에서 클러스터를 사용하는 것은 훈련 속도를 크게 증가시키지 못했다. 뿐만 아니라 수렴 성능이 나빠지거나 수렴 자체가 불가능한 경우도 발생했다. 따라서 sparse 워드 임베딩을 훈련시키는 경우, 분산 딥러닝이 큰 효용을 가지지 못하는 것으로 보인다.

REFERENCES

[1] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*(2013).

[2] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[3] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[4] Joulin, Armand, et al. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).

[5] Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2013.

[6] Levy, Omer, and Yoav Goldberg. "Linguistic regularities in sparse and explicit word representations." Proceedings of the eighteenth conference on computational natural language learning. 2014.

[7] Levy, Omer, and Yoav Goldberg. "Neural word embedding as implicit matrix factorization." Advances in neural information processing systems. 2014.

[8] Levy, Omer, Yoav Goldberg, and Ido Dagan. "Improving distributional similarity with lessons learned from word embeddings." Transactions of the Association for Computational Linguistics 3 (2015): 211-225.

[9] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[10] Isard, Michael, et al. "Dryad: distributed data-parallel programs from sequential building blocks." ACM SIGOPS operating systems review. Vol. 41. No. 3. ACM, 2007.

[11] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[12] Toshniwal, Ankit, et al. "Storm@ twitter." Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014.

[13] Zaharia, Matei, et al. "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters." HotCloud 12 (2012): 10- 10.

[14] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[15] Recht, Benjamin, et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." Advances in neural information processing systems. 2011.

[16] Chen, Jianmin, et al. "Revisiting distributed synchronous SGD." arXiv preprint arXiv:1604.00981 (2016).

[17] Ho, Qirong, et al. "More effective distributed ml via a stale synchronous parallel parameter server." Advances in neural information processing systems. 2013.

[18] Strom, Nikko. "Scalable distributed DNN training using commodity GPU cloud computing." Sixteenth Annual Conference of the International Speech Communication Association. 2015.

[19] Parallax: Automatic Data-Parallel Training of Deep Neural Networks. Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, Byung-Gon Chun. arXiv:1808.02621, August 2018.

[20] Nair, Arun, et al. "Massively parallel methods for deep reinforcement learning." arXiv preprint arXiv:1507.04296 (2015).

[21] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. 2016.

[22] Alain, Guillaume, et al. "Variance reduction in SGD by distributed importance sampling." arXiv preprint arXiv:1511.06481 (2015).

[23] Schaul, Tom, et al. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).

[24] Horgan, Dan, et al. "Distributed prioritized experience replay." arXiv preprint arXiv:1803.00933 (2018).