**데이터사이언스를 위한 머신러닝 및 딥러닝 2**

**HW3 (Self-supervised Learning)**

**서울대학교 산업공학과**

**2021-28863 유재상**

Self-supervised learning method 중 하나인 “contrastive learning”에 대해서 코드를 돌려보았습니다. 시작 하기에 앞서, 제 연구실 내의 GPU server가 존재하지 않아, google colab의 무료버전을 사용하여 코드를 돌렸습니다. 다만, 이 역시 제한된 하루의 사용량이 정해져 있어, 원했던 루프를 모두 다 돌리지 못하여, 기존에 제한된 source를 가지고 진행했음을 먼저 말씀드립니다. 그래도, Negative & positive samples 이 각각의 파라미터에 따라서 얼만큼 결과값이 달라지는 지에 대한 intuition은 얻을 수 있어 그런 점들을 본 리포트에 표현하고자 합니다.

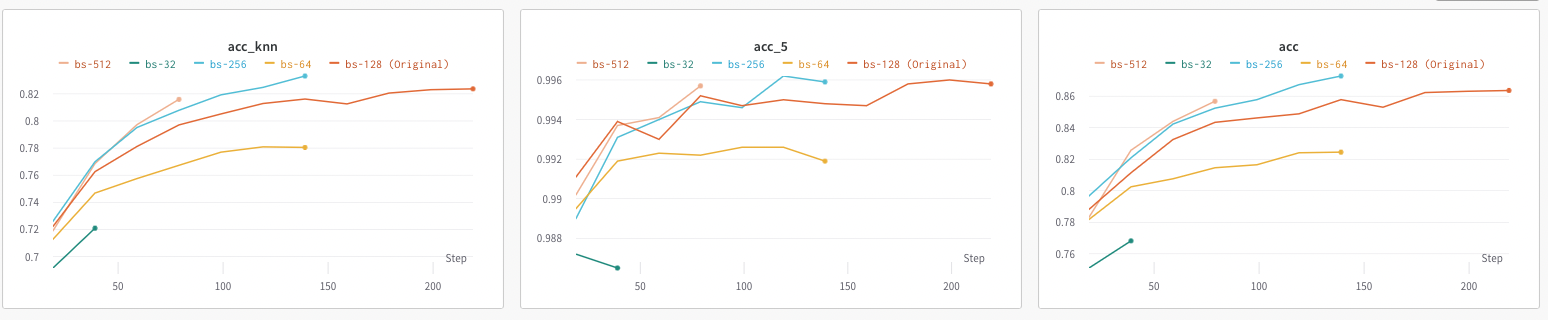
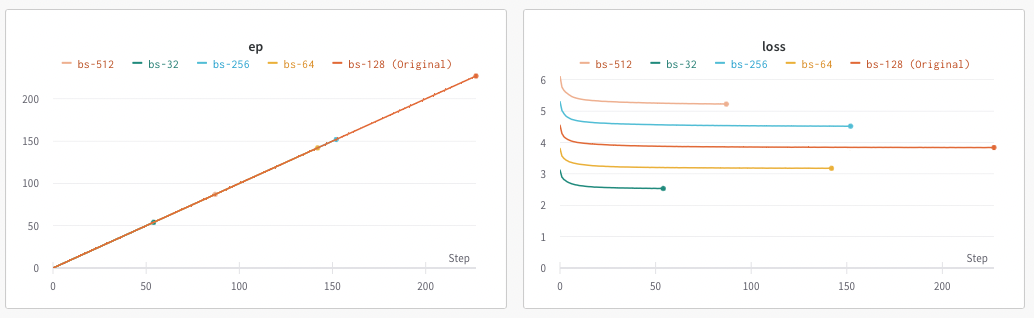
1. **Self-supervised Learning**

기존에 label을 가지고 있는 데이터를 활용하여 모델을 학습하는 것이 일반적이였으나, 모델이 점차 복잡해짐에 따라, 혹은 더 정확한 값을 예측하려고 하는 시도에 따라서, 그에 상응하는 필요로 한 데이터 수의 양이 너무나 많아졌습니다. 이 상황속에서, label이 있는 데이터는 대단히 비용이 많이 들기 때문에, label이 없는 더 큰 데이터셋을 활용하고자 하는 시도들이 시작되었습니다. 이때에 있어 Self-supervised learning은, 기존에 label을 갖고 있던 다른 문제와는 달리, untagged data를 기반으로 하여, 모델 스스로 학습 데이터에 대한 분류문제를 수행한다고 하여, 앞 단에 “self”라는 단어가 붙어졌습니다. 즉, 실제 데이터가 어떠한 범주에 해당하는지를 모르는 과정속에서 data 특징에 따라서 각자의 다른 범주로 묶는 과정을 수행해야 하는데, 이를 좀 더 자세하게 알아보고자 합니다. Self-supervised learning 내에서도 다양한 방법론이 존재하지만, 기본적으로는 두가지 (Pre-trained model and Downstream task)를 수행합니다. 이 때에, 사전 학습된 모델 생성의 경우엔, tag가 없는 상황속에서 대량의 데이터 (언어에서는 일반적인 문장)을 활용하여 다음 문장 혹은 Masking 한 단어를 예측하는 데 활용이 됩니다. 그 뒤에 소량의 tagged data를 활용하여, 앞 단의 pre trained model을 fine tuning 하는데 활용함으로써, 모델의 정확도를 높입니다. 이는 보통 기존 layer에 몇 개의 층을 더 추가하여 특성에 맞는 목적을 수행합니다.

* 1. **Contrastive Learning**

그 중, contrastive learning은 입력 샘플 간의 “비교 (대조)”를 통해, 모델을 학습하는 방법론을 의미합니다. 따라서 위에서 언급하였듯이, 사전에 정답 데이터 (label)을 구축하지 않은 classification 모델에서, 데이터 양이 많고 비용이 들지 않는 비정형데이터를 활용하는 방법을 “입력 샘플 간의 비교”를 통해 학습하고자 하는 것을 의미합니다. 더 나아가서, 이는 representation learning을 수행하기 위한 하나의 방법론으로 볼 수 있습니다. 즉, 학습된 표현 공간 내에서 “Intra 평균은 가깝게, Inter 평균은 멀게”함으로써, 표현 상에서 각 데이터의 특성에 맞게 clustering을 하고자 합니다. 또 다른 특징으로는 contrastive learning은 다른 문제를 다루고자 fine tuning 을 할 때, 모델의 아키텍쳐를 수정할 필요가 없다는 점에서 장점이 있습니다.

따라서, 비슷한 데이터를 어떻게 활용하는지에 따라 모델의 performance는 달라질 수 있고, 반대로 다른 데이터를 어떻게 활용하는지에 따라서도 모델의 performance가 달라질 수 있습니다. 본 과제 3은 바로 이 부분에 대해서 각각의 효과에 따른 performance의 차이를 보고자 합니다. 2장에서는 negative samples와 관련된 parameter를 변경함에 따른 성능 차이를 보여주고, 3장에서는 positive samples와 관련된 parameter를 변경함에 따른 성능 차이를 보여주겠습니다.

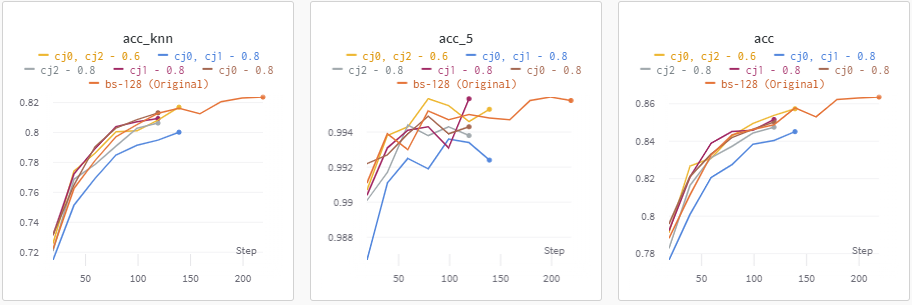
1. **Negative samples (with different parameters : batch size (bs))  
   ** 먼저, negative sample의 수의 변화에 따른 모델의 성능을 비교해보겠습니다. Negative samples의 변화를 줄 수 있는 방법은 batch size를 통해 얻을 수 있는데, 가령 사진 속에서 특정 대상이 나타내는 공간을 제외하고는 나머지 부분은 관련이 없는 부분으로 고려할 수 있습니다. 따라서 사진의 크기, 즉 batch size를 어떻게 하는지에 negative sample의 수를 변경할 수 있고 이에 따른 효과를 이제 각 그래프를 통해 보고자 한다.   
     
    앞서 언급 드린 대로, 모델을 돌릴 수 있는 환경이 충분하지 않음에 따라, 중간에 GPU 사용 제한이 나오기 전까지만 돌린 내용을 고려하였습니다. 이에 각 parameter 변경에 따른 epoch수가 다르기 때문에, 그래프의 x의 정도가 다를 수 있습니다. 그것을 제외하더라도 accuracy 부분을 봤을 경우, batch\_size를 크게 함에 따라서, 모델의 정확도는 증가함을 볼 수 있습니다. 본 실험에서는 batch\_size를 크게 5가지 [32,64,128,256,512]를 사용하였고, accuracy , acrruacy + KNN , accuracy + 5 candidates 측면모두 배치사이즈가 컸을 때 높은 성과를 보이고 있다. 즉, 이는 우리가 반대로 생각할 수 있는 반례 케이스들을 많이 학습에 넣음에 따라서, 모델로 하여금 본인과 차이가 나는 부분이 어떤 점임을 충분히 학습할 수 있게 됩니다. 하지만, 32개의 배치를 하게 되었을 시엔, 31개의 negative samples를 활용하기 때문에 모델이 충분히 그러한 차이를 학습하지 못했다고 생각할 수 있습니다.****

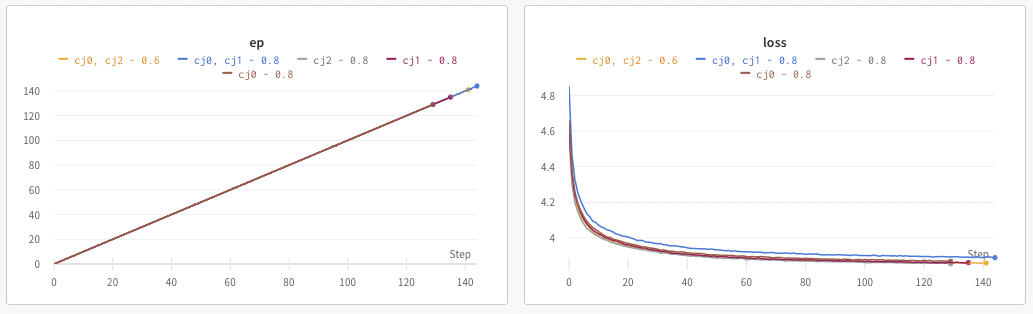
앞서 언급 드렸듯이, epoch의 수는 각 모델에 따라 다르게 나타납니다. 이는 전체 제가 하루에 사용할 수 있는 리소스가 동일할 때, 배치 사이즈에 따라 그에 해당하는 epoch도 달라지기 때문입니다. 하지만 그것보다 더 중요한 것은 loss 측면에 있습니다. Loss에 대한 내용이 바로, positive pair의 embedding은 최대한 가깝게 처리하고 negative pair의 embedding은 최대한 멀게 하는 부분을 직접적으로 수행하는 부분입니다. Batch size가 증가함에 따라, N-1개의 negative samples 와 1개의 positive sample을 갖고 있는 상황속에서 “N-pair loss’의 값이 커지게 됩니다. 따라서 이와 관련된 부분을 잘 보여주고 있다고 볼 수 있습니다.

**테이블이(가) 표시된 사진

자동 생성된 설명**

그 외의 기타 GPU 및 memory 사용량에 따른 그래프를 보여주고 있으나, 그렇게 큰 차이는 보이지 않고 있습니다. 뚜렷한 차이가 없기 때문에 이 부분 보다는 앞에서의 loss, accuracy 부분에 대해서 보다 더 자세히 분석할 필요가 있습니다.

1. **Positive samples (with different data augmentation parameter)   
   **

Positive samples의 경우엔 보통 data augmentation을 활용하여 다룰 수 있는데, 데이터증강방법은 크게 2가지로 나눌 수 있습니다. 첫번째는 기존 positive 한 data를 알고 있다고 생각했을 경우, 원본에 noise를 추가함으로써 변형을 주는 방법이고, 두번째는 본 데이터의 색감, 대조, hue 등의 degree 자체를 바꾸거나, 이러한 transform의 sequence를 바꿀 수 있습니다. 제가 활용한 방법은 후자입니다. 일단 parameter 하나를 변경한 경우엔 기존 sequence를 유지한 채 degree만 바꾸었고, 두개의 parameter를 변경한 경우엔 그 안에서의 sequence 또한 변경하여서 표현했습니다. 먼저 accuracy를 봤을 경우, 기존 bs 128를 다룬 benchmark performance와 비교했을 때, 무조건적으로 어떠한 결과값을 가질 것이다 라는 것을 예상하기 어렵습니다. Negative sample의 경우엔 batch size를 증가함에 따라서 그에 대한 값의 차이가 확연하게 차이가 났으나, 제가 예시로 잡은 positive samples의 경우엔, 변형할 수 있는 parameter 수 (= j0, j1, j2, j3) + sequence 등을 고려할 때 너무나 많은 경우의수가 존재하여서 비교하기 어려웠습니다. 또한, 특정 한 parameter의 degree만 0.3, 0.5, 0.7 등으로 비교하는게 아니라, 각각의 파라미터를 단순 benchmark parameter보다 높였을 경우에만 판단하였기에 정확한 비교가 될 수 있다고는 생각하지 않습니다. 다만, 데이터를 증강할 때에 어떤 sequence와 어떤 parameter degree를 하는지에 따라서 “모델이 비슷함을 판단하는 근거”가 달라질 수 있다는 점은 확연하게 볼 수 있습니다. 따라서, positive samples 를 통해 모델의 성능 및 정확도를 높이기 위해서는 각 변수의 degree 뿐만 아니라, transform 하는 순서 및 architecture의 중요성을 언급 할 수 있겠습니다. 이는 Neural network를 만들 때와 같이 각각의 hyperparameter 및 batch-norm & skip connection 의 순서에 따른 모델의 성능차이가 날 수 있음과 동일하게 해석할 수 있겠습니다. ****

Epoch는 negative와 달리 비슷한 step에 멈춰 있음을 볼 수 있고, 이는 앞단에서 batch size에 따른 할당된 자원 사용과는 달리, 거의 동일한 resource를 사용함으로 이해를 할 수 있습니다. Loss 역시 위의 accuracy를 다룬 바와 같이, 어떤 parameter를 변경할지, 그 parameter의 degree는 어느 정도로 할지, 그리고 전체의 transform procedure는 어떻게 되는지 등에 따라서 달라질 수 있음을 보여주는 그림으로 이해할 수 있습니다.

아래의 GPU 및 memory 사용량 역시, 각 모델에 따라서의 결과가 완전 상이하기 보다는 비슷한 추세를 보이고 있어, 그리 눈에 띄는 큰 차이를 보여주기엔 한계가 있다고 생각합니다.

**테이블이(가) 표시된 사진

자동 생성된 설명**