Swivel: submatrix-wise vector embedding learner

0. Reference Paper

Swivel: Improving Embeddings by Noticing What's Missing, Noam Shazeer et al.

1. Basic Setup

notation을 정리하자.

- $\mathbf{X}$: $m \times n$ co-occurrence matrix, where each cell x_{ij} is the observed co-occurrence count of row feature i with column feature j.
- $x_{i*} = \sum_j x_{ij}$: marginal counts of each row feature
- $x_{*j} = \sum_i x_{ij}$: marginal counts of each column feature
- $|D| = \sum_{i,j} x_{ij}$: overall sum of all the cells in the matrix

co-occurrence matrix $\mathbf{X}$의 행들을 feature의 빈도에 따라 내림차순으로 정리하고 k개의 원소를 가지는 row blocks로 나눈다. 여기서 k는 computational efficiency에 기반하여 결정된다. 총 m개의 행에 대하여 k개의 원소를 가지는 row blocks로 나누므로, 총 m/k개의 row blocks가 생긴다. 각 row blocks의 원소들은 m/k개의 잘리는 지점에 따라 배정된다. 예를 들어 총 100개의 행이 있고 $k = 10$으로 잡으면 매 10번째 행이 한 row block으로 지정된다. 즉, 첫 번째 row block은 (0, 10, 20, ...)이고 두 번째 row block은 (1, 11, 21, ...) 이런 식으로 결정된다. 맨 처음에 $\mathbf{X}$의 행을 내림차순으로 배열했기 때문에 각 row block에는 상위 행과 하위 행이 모두 포함된다. 이와 같은 과정을 열에 대해서도 동일하게 하여 n/k개의 column blocks을 만든다.
이제 m/k개의 row blocks와 n/k개의 column blocks에서 각 쌍 $(i, \ , j)$에 대해 $k \times k$ submatrix인 χ_{ij}을 만든다. (row blocks와 column blocks는 k개의 원소를 가지므로 $k \times k$ submatrix) χ_{ij}의 구체적인 원소를 살펴보면 아래와 같다.

$$\chi_{ij} = \begin{bmatrix} x_{i,\,j} & x_{i+\frac{m}{k},\,j} & \cdots & x_{i+(k-1)\frac{m}{k},\,j} \\ x_{i,\,j+\frac{n}{k}} & & & \\ \vdots & & \ddots & \\ x_{i,\,j+(k-1)\frac{n}{k}} & & & \end{bmatrix}$$

또한 나올 수 있는 χ_{ij}의 개수는 $\frac{m}{k} \times \frac{n}{k} = \frac{mn}{k^2}$개이다. 대부분은 0의 값을 가질 것이다.

2. Training

훈련 전에, 두 개의 d차원 벡터가 랜덤하게 초기화 된다. $\mathbf{W} \in \mathbb{R}^{m \times d}$와 $\mathbf{W} \in \mathbb{R}^{n \times d}$는 각각 m개의 row features에 대한 embeddings(타겟 단어들에 대한 임베딩), n개의 column features에 대한 embeddings(문맥 단어들에 대한 임베딩)이다. 훈련은 아래와 같이 진행된다.

1. χ_{ij}가 랜덤하게 선택된다. 이 말은 χ_{ij}를 구성하는 i번째 row block과 j번째 column block도 같이 선택된다는 뜻이다. 각각은 k개의 원소로 구성된다고 앞서 언급했다. 즉, i번째 row block으로부터 k개의 row embedding vectors을 가지는 $\mathcal{W}_i \in \mathbf{W}$와 j번째 column block으로부터 k개의 column embedding vectors을 가지는 $\tilde{\mathcal{W}}_j \in \mathbf{W}$가 선택된다.

2. $\mathcal{W}_i\tilde{\mathcal{W}}_j^T$가 계산되는데, 이는 k^2개의 predicted PMI 값이다. 이는 χ_{ij}의 observed PMI 값이랑 비교된다.

3. 이 두 값의 사이가 gradient을 계산하는데 사용된다. 이것이 각 row와 column에 대해서 쌓인다.

각 submatrix인 χ_{ij}가 따로 고려되지만 embedding parameters인 $\mathcal{W}_i$는 동일한 row block에서 공유되고 마찬가지로 $\tilde{\mathcal{W}}_j$도 동일한 column block에서 공유된다. 모수를 central parameter server에 저장함으로써 다른 worker machines에서 평행하게 훈련을 진행할 수 있다고 한다. 마치 GPU 같은 느낌인데, 역시나 논문에서도 GPU를 사용하여 높은 성능을 보였다고 하고 환경마다 다르겠지만 $k = 4096$에서 좋은 결과를 얻었다고 한다.

3. Training Objective and Cost Function

Swivel은 observed PMI를 row feature i와 column feature j의 내적 값인 $w_i^T\tilde{w}_j$으로 근사한다. 또한 co-occurrence 값이 0인 것들을 따로 처리하기 위해서 piecewise loss function을 정의한다.

Observed co-occurrence

$x_{ij} > 0$에 대해서, $w_i^T\tilde{w}_j$가 pmi$(i; j)$을 정확히 추정하길 바란다. Swivel은 이 둘 간의 weighted squared error 아래와 같이 계산한다.

$$\begin{aligned}\mathcal{L}_1(i, j) &= \frac{1}{2}f(x_{ij})\left(w_i^T\tilde{w}_j - pmi(i; j)\right)^2 \\ &= \frac{1}{2}f(x_{ij})\left(w_i^T\tilde{w}_j - log\, x_{ij} - log|D| + log\, x_{i*} + log\, x_{*j}\right)^2\end{aligned}$$

$$where\, pmi(i; j) = log\frac{P(i, j)}{P(i)P(j)} = log\frac{x_{ij}/|D|}{(x_{i*}/|D|)(x_{*j}/|D|)} = log\frac{x_{ij}\cdot|D|}{x_{i*}x_{*j}}$$

목적식 $\mathcal{L}_1$을 최소화하도록 훈련을 진행한다는 것은, weighted error sum을 최소화한다는 것이며 이는 곧 predicted pmi인 $w_i^T\tilde{w}_j$와 observed pmi인 pmi$(i; j)$을 최소화한다는 뜻이다.
$\mathcal{L}_1$을 자세히 살펴보자. $f(x_{ij})$는 co-occurrence을 뜻한다. 단어 i, j가 동시에 많이 나타날수록 이 값이 클 것이고 $\mathcal{L}_1$ 값이 커질 것이다. 그렇다면 전체적인 loss도 커질텐데, 이럴수록 $w_i^T\tilde{w}_j$와 observed pmi 값은 가까워야지 loss을 더 줄일 것이다. 다시 말해서 단어 i, j가 많이 등장할수록 강제로 두 단어에 해당하는 벡터의 내적이 pmi 값과 비슷해지도록 만드는 것이다.

Unobserved Co-occurrence

하지만 corpus에서 $|D|$는 매우 클 것이기 때문에 co-occurrence matrix는 0의 값을 마니 가지는 sparse matrix일 가능성이 크다. 즉, x_{ij} 값이 0일 수도 있는데, 이러한 경우에는 $\mathcal{L}_1$가 $-\infty$로 발산하는 문제가 발생한다.

x_{ij}를 sample로 생각하면, 어떠한 상황에서 이렇게 $x_{ij} = 0$으로 관측하는지 생각해보면 된다. 두 단어가 censored data, Cox regression과 같이 희귀한 단어라면 단지 가지고 있는 corpus의 데이터가 많지 않기 때문에 두 단어의 co-occurrence가 0일 수 있다. 만약 생존분석 관련 데이터라면 이 두 단어의 co-occurrence는 0이 아닐 것이다. 하지만 두 단어가 흔함에도 불구하고 co-occurrence가 0이라면 데이터 양의 문제가 아니다. 그냥 이 둘이 관련이 없다고 생각하는 것이 논리적이다. 어찌됐든 두 경우 모두에 대해서 모델이 pmi를 over-estimate하지 않기를 바라고 추정치에 대한 upper bound을 설정한다.
이를 $x_{ij} = 1$이라고 둠으로써 pmi 값을 smoothing하여 해결한다. 또한 smoothed pmi 값에 대해서 over-estimation하는 것에 penalty을 주는 비 대칭 cost function을 사용한다. 이는 아래 soft hinge cost function에서 구현할 수 있다.

$$\begin{aligned}\mathcal{L}_0(i,\, j) &= log\left[1 + exp\left(w_i^T \tilde{w}_j - pmi^*(i;\, j)\right)\right] \\ &= log\left[1 + exp\left(w_i^T \tilde{w}_j - log|D| + log\, x_{i*} + log\, x_{*j}\right)\right]\end{aligned}$$

여기서 pmi^*는 $x_{ij} = 0$인 co-occurrence에 대해서 $x_{ij} = 1$로 대체한 smoothed pmi 계산을 의미한다. $\mathcal{L}_0$을 자세히 살펴보자. 우선, $\mathcal{L}_0$은 두 단어의 co-occurrence가 0일 때 해당되는 손실 함수임을 다시 한번 상기하자. 단어 i, j가 흔한 단어라면 x_{i*}, x_{*j}는 클 것이다. 따라서 $\mathcal{L}_0$을 줄이기 위해서 $w_i^T \tilde{w}_j$가 작아지거나 심지어 음수가 될 수도 있다. 따라서 단어 i, j간에 anti-correlation을 잡아내는 것이다. 반대로 두 단어가 희귀하다고 하자. 따라서 x_{i*}, x_{*j}도 작다. $\mathcal{L}_0$의 마지막 term인 $log\, x_{i*}, log\, x_{*j}$도 작아지므로 $\mathcal{L}_0$을 최소화하는데, 위 경우처럼 무조건 작아져야 한다는 강제는 없다 (물론 심각한 penalty을 발생시키지 않는다는 조건 하에서)
이런 식으로 soft hinge loss function은 co-occurrence가 0인 두 단어 간에 사람이 할수 있는 생각을 반영한다.