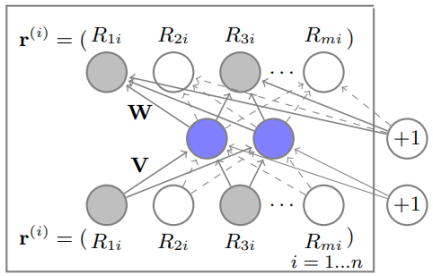
AutoRec & ConvMF

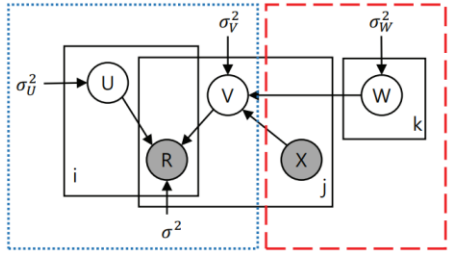
박상민

Contents



CONTENTS A

AutoRec



CONTENTS B

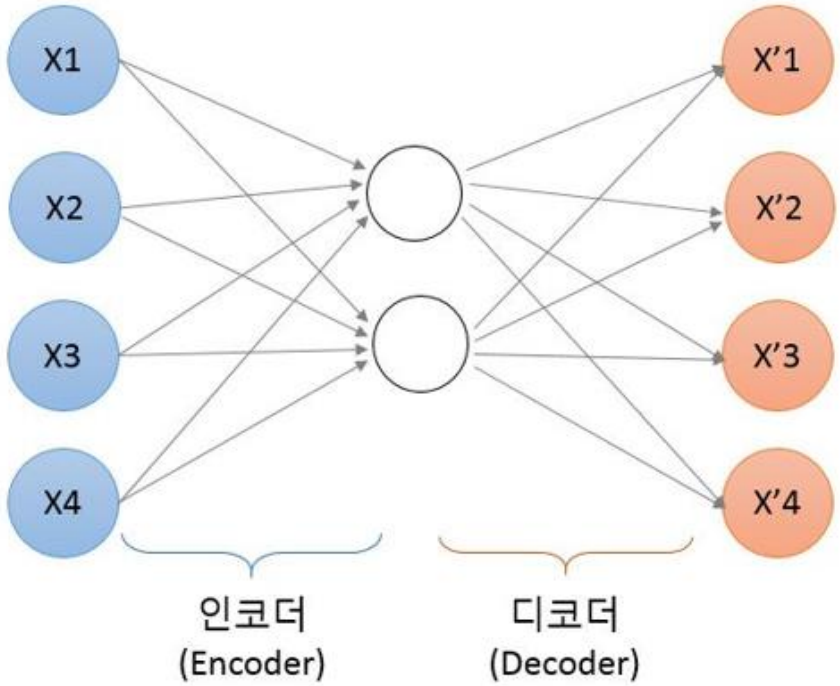
ConvMF

```
mask = x > 0
pred = model(x)
loss = torch.mean(((x - pred)[mask])**2)
loss.backward()
optimizer.step()
losses.append(np.sqrt(loss.item()))
```

CONTENTS C

Implementation

AutoRec – Preliminaries



AutoEncoder

- 동일한 형태의 입/출력 구조.
- Encoder 단계에서는 본래 input으로부터 lower dimensional latent feature를 encode한다.
- Decoder 단계에서는 위 latent feature를 input과 같은 크기의 output을 input과 가능한 유사하게 산출.

AutoRec – Details

$$\min_{\theta} \sum_{\mathbf{r} \in \mathbf{S}} ||\mathbf{r} - h(\mathbf{r}; \theta)||_2^2,$$

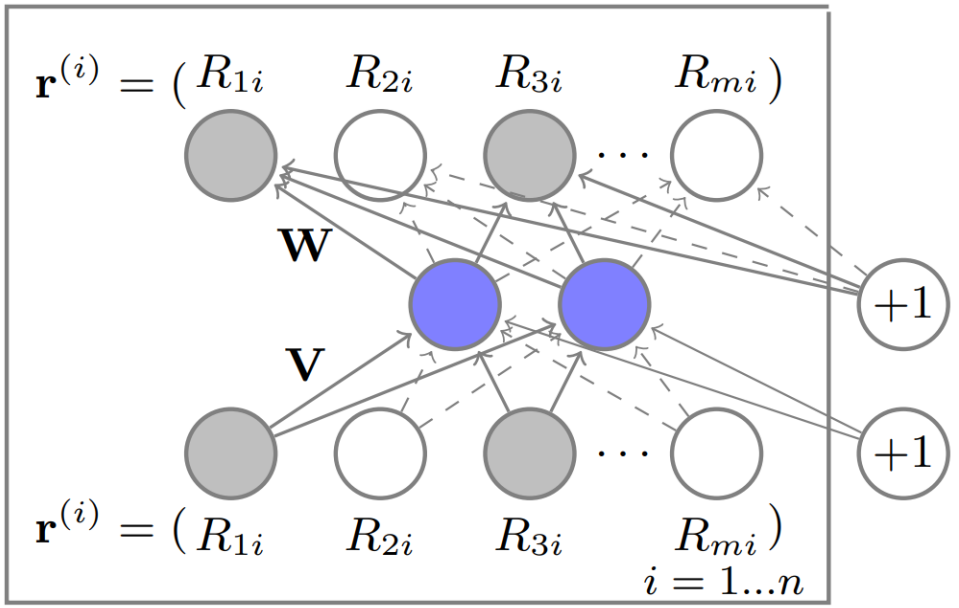
AutoRec : AutoEncoder + Recommender system

- R_i(item i에 대해 user들이 매긴 rating 점수들) 기반 (R_u로도 동일하게 적용가능)
- R_i가 encode, decode 단계를 거쳐 reconstruct 된 것이 h 항.
- Reconstruct 됐을 때 missing ratings에 대해 predict하도록 한다.

AutoRec – Details

$$h(\mathbf{r};\theta) = f\left(\mathbf{W} \cdot g(\mathbf{V}\mathbf{r} + \boldsymbol{\mu}) + \mathbf{b}\right)$$

$$\min_{\theta} \sum_{i=1}^{n} ||\mathbf{r}^{(i)} - h(\mathbf{r}^{(i)};\theta))||_{\mathcal{O}}^2 + \frac{\lambda}{2} \cdot (||\mathbf{W}||_F^2 + ||\mathbf{V}||_F^2)$$



AutoRec

- W,V 는 transformation, μ와 b는 bias, f와 g는 activation function을 의미.
- Observed data에 대해서만 back propagation을 통한 update가 이루어진다.
- Overfitting을 방지하고자하는 Regularization term.
- 2mk + m + k 개의 parameters

AutoRec – Details

AutoRec's distinct feature from others

- MLE가 아닌 RMSE사용
- Gradient back propagatation을 이용한 비교적 빠른 computation
- Parameter 수 감소 (mkr >> 2mk+m+k)
- Item / user 중 택일로 embedding 하여 latent space 생성.
- Non-linear latent representation 학습가능.

AutoRec – Experiment & Conclusion

	ML-1M	ML-10M
U-RBM	0.881	0.823
I-RBM	0.854	0.825
U-AutoRec	0.874	0.867
I-AutoRec	**0.831**	**0.782**

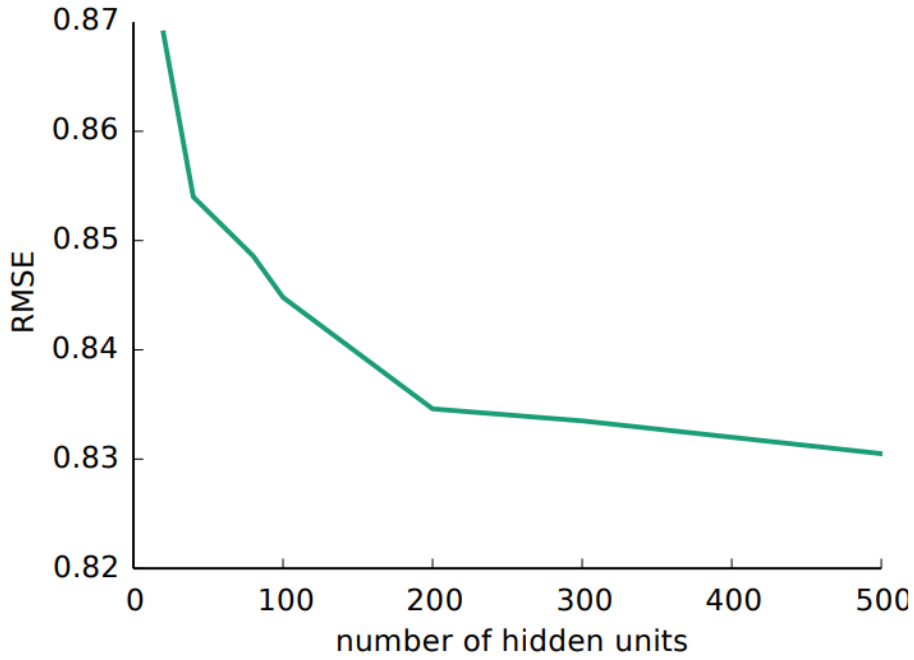
$f(\cdot)$	$g(\cdot)$	RMSE
Identity	Identity	0.872
Sigmoid	Identity	0.852
Identity	Sigmoid	**0.831**
Sigmoid	Sigmoid	0.836

Item based vs User based / non-linear act_f

- Item latent feature based model 이 User based model보다 좋은 성능을 보였다.
- Non-linear act_f이 실제로 성능 개선에 기여를 하는 모습을 보였다. (MF 대비 장점)

AutoRec – Experiment & Conclusion

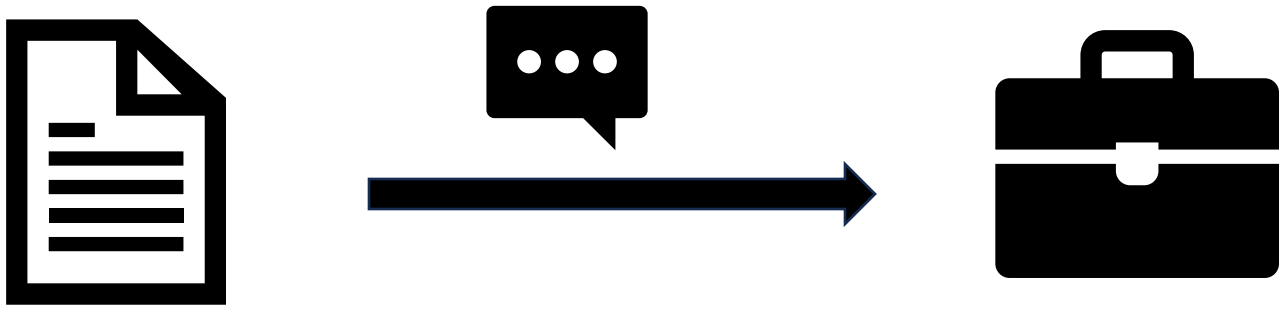
	ML-1M	ML-10M	Netflix
BiasedMF	0.845	0.803	0.844
I-RBM	0.854	0.825	-
U-RBM	0.881	0.823	0.845
LLORMA	0.833	**0.782**	0.834
I-AutoRec	**0.831**	**0.782**	**0.823**



AutoRec's outperformance / hidden unit impact

- 기존의 다른 competitors보다 전반적으로 우월한 성능을 보였다.
- Hidden units가 증가함에 따라 성능이 향상되는 것이 확인 되었다.
- Layer를 3개로 더 deep하게 쌓았을 때 미약하게나마 성능이 향상 되었다.

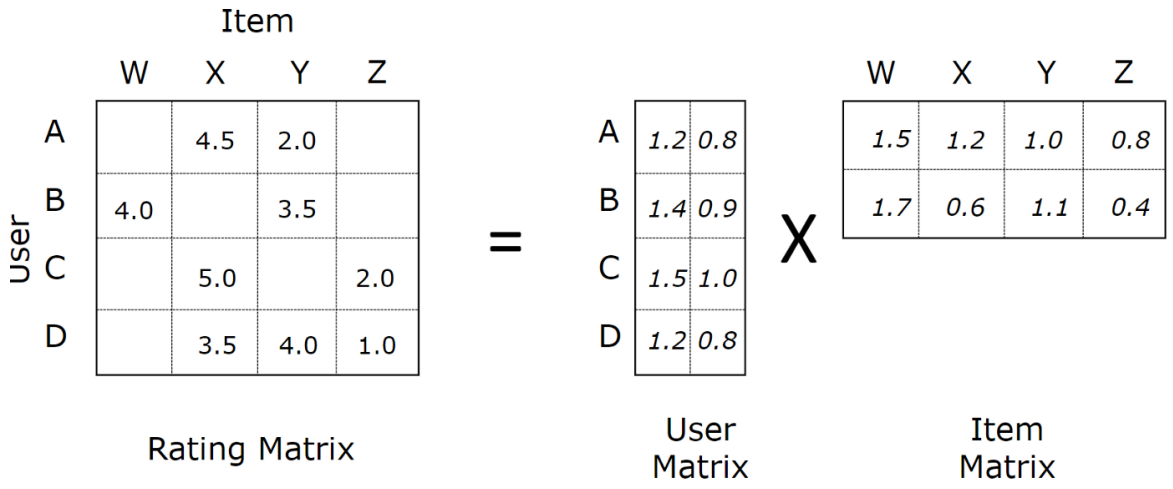
ConvMF – Preliminaries



BOW(Bag of words)

- 단어들의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도에만 집중하는 텍스트 데이터의 수치화 방법
- LDA, SDAE 와 같이 document description을 추가적으로 활용하던 기존 기법들이 이 BOW방식을 채택.

ConvMF – Preliminaries

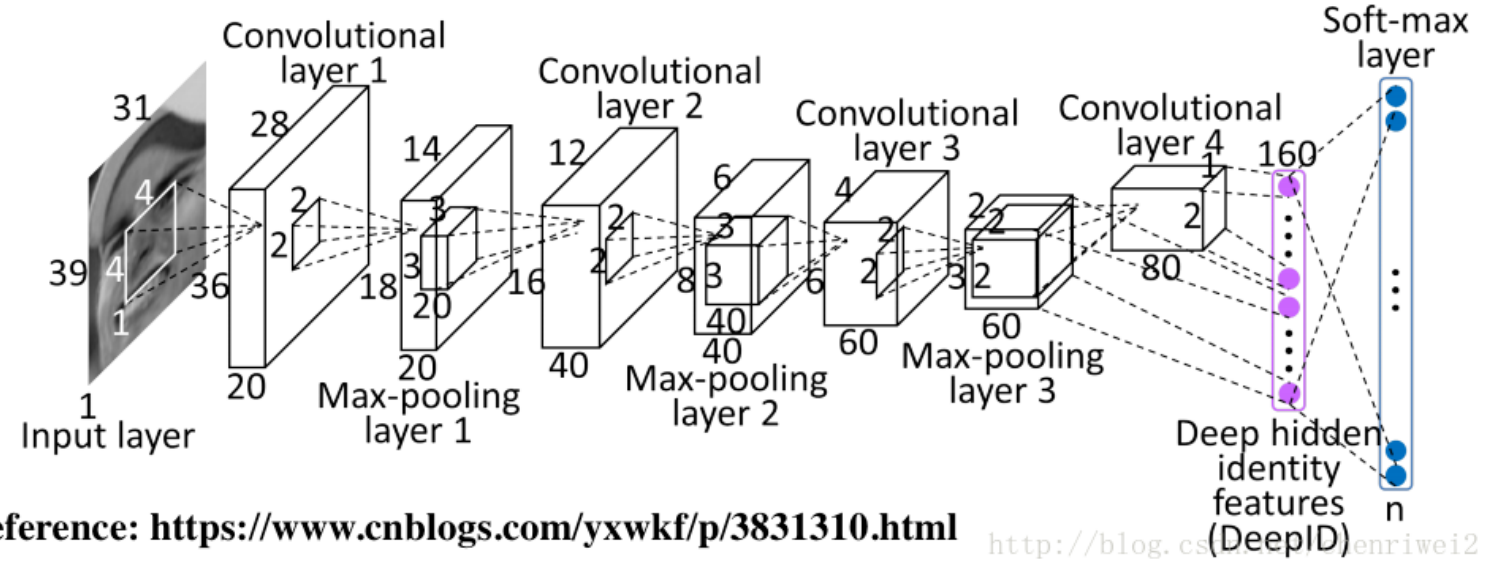


$$\mathcal{L} = \sum_{i}^{N} \sum_{j}^{M} I_{ij}(r_{ij} - u_i^T v_j)^2 + \lambda_u \sum_{i}^{N} \|u_i\|^2 + \lambda_v \sum_{j}^{M} \|v_j\|^2$$

MF(Matrix Factorization)

- User x Item matrix를 latent feature로 User matrix, Item matrix로 분해

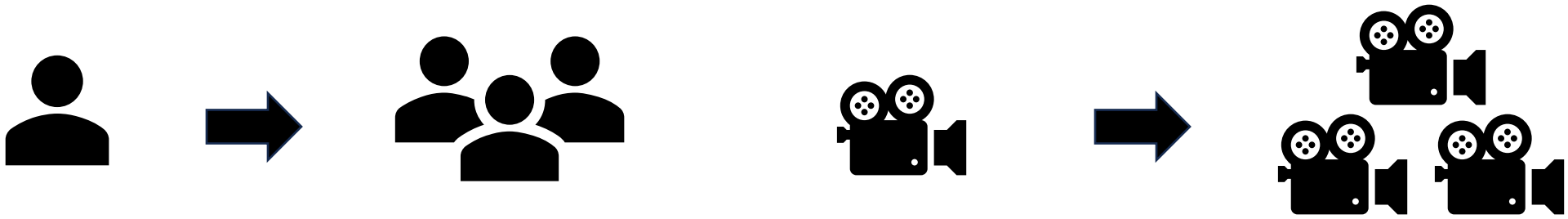
ConvMF – Preliminaries



CNN(Convolutional Neural Network)

- filter를 slide해가면서 **local feature**를 담는 convolutional layer를 형성한다.
- Max pooling으로 concise한 data representative한 feature를 뽑아낸다.

ConvMF - Intro



users, # items 급증

- Sparseness 역시 함께 증가
- Rating 이외 정보도 활용해보자!

  Ex) Social network (SoReg,SoRec) , Item discription(LDA, SDAE)

ConvMF - Intro

"People trust the man." vs "People betray his trust finally."

기존의 document information integrated model은 **BOW** 방식 활용

>> **Contextual information을 무시**한다.

>> Deeper understanding of document description 은 Rating prediction accuracy에 도움을 주지 않을까?

CNN + PMF

CNN

- Document의 **local feature** capture : Contextual information 획득.
- Rating 수가 **적어서** description에 더 **의존**해야할 때 영향이 더 클 것이다.
- BOW 방식이 아니기에 **pre-trained word embedding** 역시 사용 가능.
- 하지만 CNN은 recommendation에 **적합하지는 않다.**
- **PMF와 접목시키자!**

ConvMF - Details

$$p(R|U,V,\sigma^2) = \prod_i^N \prod_j^M N(r_{ij}|u_i^T v_j, \sigma^2)^{I_{ij}}$$

$$p(U|\sigma_U^2) = \prod_i^N N(u_i|0, \sigma_U^2 I)$$

$$v_j = cnn(W, X_j) + \epsilon_j$$

$$\epsilon_j \sim N(0, \sigma_V^2 I)$$

$$p(V|W,X,\sigma_V^2) = \prod_j^M N(v_j|cnn(W,X_j), \sigma_V^2 I)$$

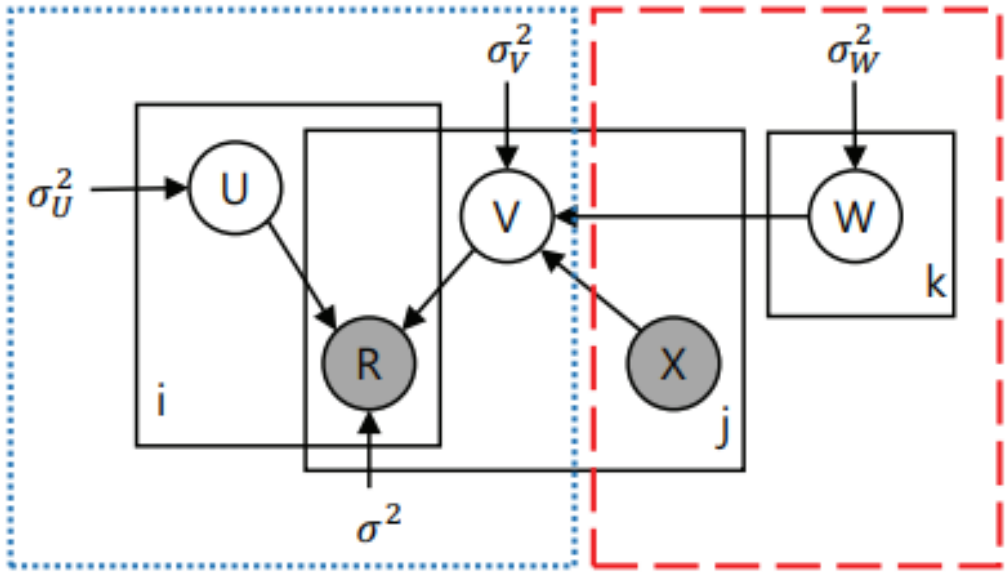


Figure 1: Graphical model of ConvMF model: PMF part in left (dotted-blue); CNN part in right (dashed-red)

W : weight for CNN

X_j : document for item j

Epsillon_j : Gaussian noise

ConvMF - Details

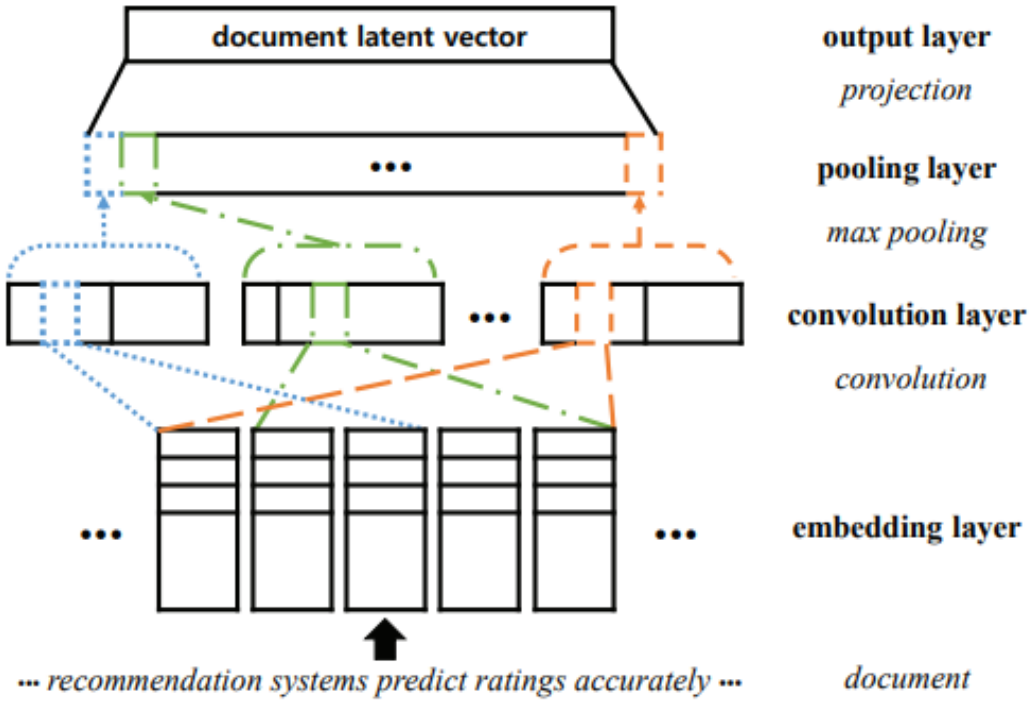
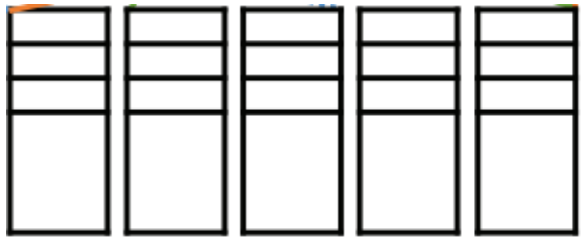


Figure 2: Our CNN architecture for ConvMF

ConvMF - Details

$$D = \begin{bmatrix} \cdots & | & | & | & \cdots \\ & w_{i-1} & w_i & w_{i+1} & \\ & | & | & | & \end{bmatrix}$$



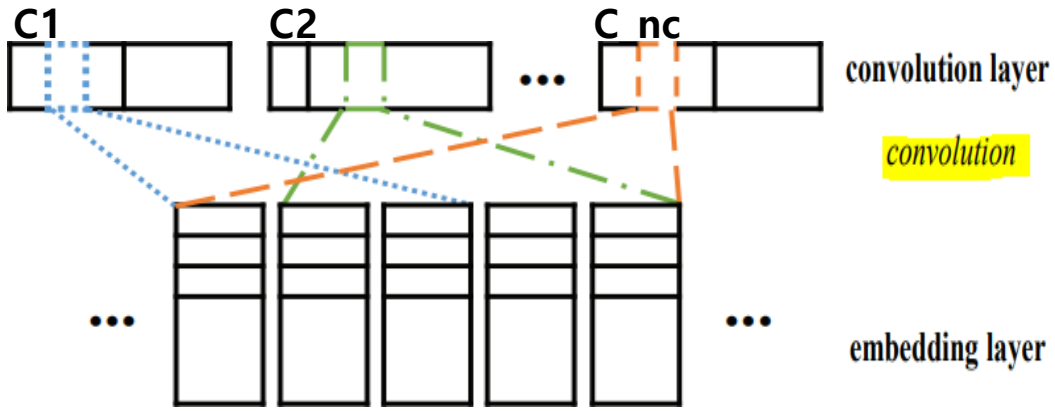
Embedding Layer

- Raw document > dense numeric matrix
- L개의 단어들을 size(dim) p로 embedding.
- Random or Pre-trained word embedding(Glove)

ConvMF - Details

$$c_i^j = f(W_c^j * D_{(:,i:(i+ws-1))} + b_c^j)$$

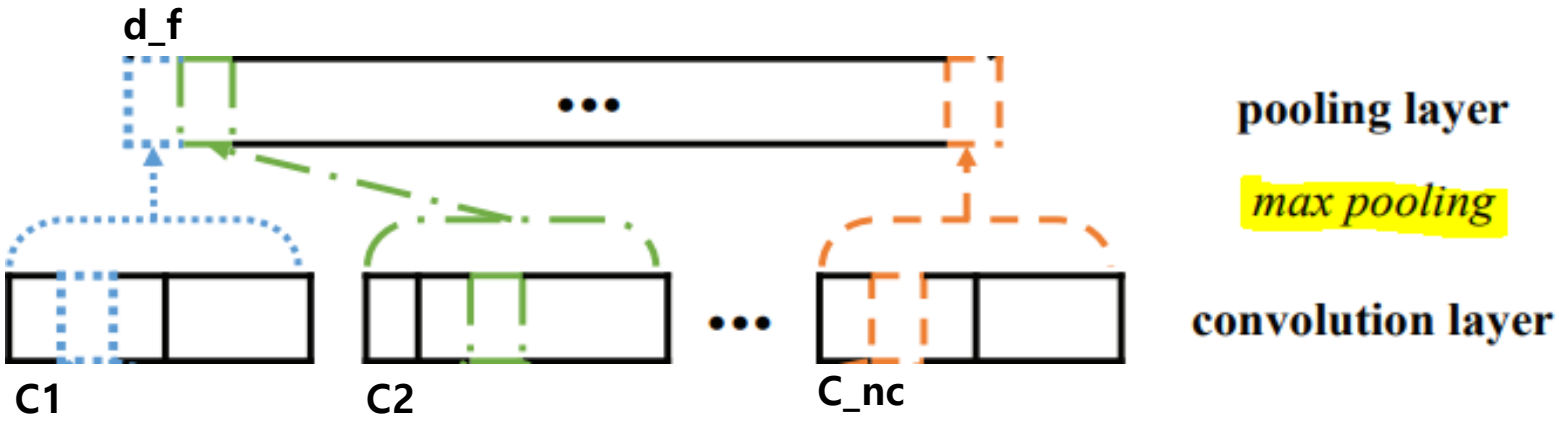
$$c^j = [\, c_1^j,\ c_2^j,\ \ldots,\ c_i^j,\ \ldots,\ c_{l-ws+1}^j]$$



Convolutional Layer

- Contextual feature 뽑아낸다.
- C_ij = ith word가 shared weight W_cj에 의해 추출된 Contextual feature (window size : ws)
- Window size는 surrounding words의 개수. (Filter의 너비와 유사)
- C_j = item j의 document 에 대해 weight W_c를 적용하여 얻은 contextual feature vector
- 서로다른 W를 적용하여 n_c개의 contextual feature vector를 얻는다.

ConvMF - Details

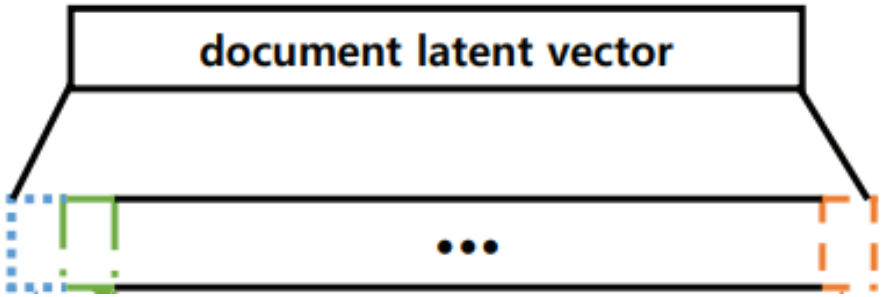


$$d_f = [\max(c^1), \max(c^2), \ldots, \max(c^j), \ldots, \max(c^{n_c})]$$

Pooling Layer

- Max pooling으로 Representative features 뽑아낸다.
- **Variable**(ws) length > **fixed-length** feature vector
- 쓸데없이 많은(도움이 거의 되지 않으며) contextual feature 중 representative한 것들만 추출

ConvMF - Details



$$s = \tanh(W_{f_2}\{\tanh(W_{f_1} d_f + b_{f_1})\} + b_{f_2})$$

$$s_j = cnn(W, X_j)$$

Output Layer

- dim k로 projection하여 recommendation task에 적합하게 한다.
- Document latent vector S_j for item j

ConvMF - Details

$$\max_{U,V,W} p(U,V,W|R,X,\sigma^2,\sigma_U^2,\sigma_V^2,\sigma_W^2)$$

$$= \max_{U,V,W} [p(R|U,V,\sigma^2)p(U|\sigma_U^2)p(V|W,X,\sigma_V^2)p(W|\sigma_W^2)]$$

$$\mathcal{L}(U,V,W) = \sum_i^N \sum_j^M \frac{I_{ij}}{2}(r_{ij} - u_i^T v_j)_2 + \frac{\lambda_U}{2}\sum_i^N \|u_i\|_2$$

$$+ \frac{\lambda_V}{2}\sum_j^M \|v_j - cnn(W, X_j)\|_2 + \frac{\lambda_W}{2}\sum_k^{|w_k|} \|w_k\|_2,$$

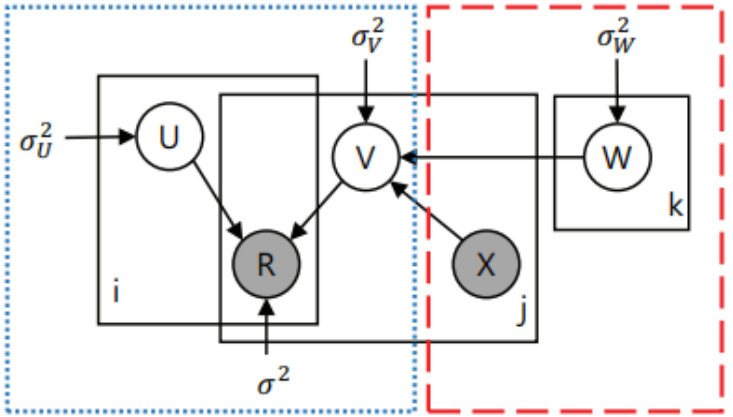


Figure 1: Graphical model of ConvMF model: PMF part in left (dotted-blue); CNN part in right (dashed-red)

Optimization Methodology

- 앞선 과정을 통한 map 식 변형
- Negative logarithm으로 Loss function 도출
- W regularization term 추가 및 spherical이 아닌 V regularization term

ConvMF - Details

$$u_i \leftarrow (VI_iV^T + \lambda_U I_K)^{-1} VR_i$$

$$v_j \leftarrow (UI_jU^T + \lambda_V I_K)^{-1} (UR_j + \lambda_V cnn(W, X_j))$$

Coordinate descent for U, V

- 한 Variable을 제외하고 나머지는 fix한채 optimize한다.
- U : V와 W를 constant / V : U 와 W를 constant 취급

ConvMF - Details

$$\mathcal{E}(W) = \frac{\lambda_V}{2} \sum_{j}^{M} \|(v_j - cnn(W, X_j))\|^2 + \frac{\lambda_W}{2} \sum_{k}^{|w_k|} \|w_k\|^2 + \text{constant}$$

Back propagation for W

- W는 CNN에서 max pooling, non-linear act function을 거쳤기에 coordinate descent적용 불가.
- 대신 Loss function을 단순히 W에 대한 Square error function에 지나지 않는다.
- U와 V는 constant로 바라본다.

ConvMF - Details

$$
\begin{aligned}
r_{ij} &\approx \mathbb{E}[r_{ij}|u_i^T v_j, \sigma^2] \\
&= u_i^T v_j = u_i^T (cnn(W, X_j) + \epsilon_j)
\end{aligned}
$$

ConvMF - Details

$$O(k^2 n_R + k^3 N + k^3 M)$$

$$O(n_c \cdot p \cdot l \cdot M)$$

$$O(k^2 n_R + k^3 N + k^3 M + n_c \cdot p \cdot l \cdot M)$$

Time Complexity Analysis

- Coordinate descent를 진행한 U,V
- CNN computation cost가 dominate하는 W
- Filter / Embedding dim / Words / Items

ConvMF - Experiment

Dataset	# users	# items	# ratings	density
ML-1m	6,040	3,544	993,482	4.641%
ML-10m	69,878	10,073	9,945,875	1.413%
AIV	29,757	15,149	135,188	0.030%

Table 1: Data statistic on three real-world datasets

MovieLens & Amazon

- Amazon은 제품 리뷰, MovieLens는 IMDB plot을 document로 이용
- Max length, stop words removal, tf-idf score etc..
- Amazon dataset은 특히나 더 sparse한 양상을 보인다.

ConvMF - Experiment

Model	ML-1m		ML-10m		AIV	
	λ_U	λ_V	λ_U	λ_V	λ_U	λ_V
PMF	0.01	10000	10	100	0.1	0.1
CTR	100	1	10	100	10	0.1
CDL	10	100	100	10	0.1	100
ConvMF	100	10	10	100	1	100
ConvMF+	100	10	10	100	1	100

Table 2: Parameter Setting of λ_U and λ_V

$$\text{RMSE} = \sqrt{\frac{\sum_{i,j}^{N,M}(r_{ij} - \hat{r}_{ij})^2}{\#\text{ of ratings}}}$$

Competitors & Evaluation Protocol

- CTR/CDL은 LDA/SDAE 활용한 recommendation model
- ConvMF+는 pre-trained word embedding 이용.

ConvMF - Experiment

	Dataset		
Model	ML-1m	ML-10m	AIV
PMF	0.8971	0.8311	1.4118
CTR	0.8969	0.8275	1.5496
CDL	0.8879	0.8186	1.3594
ConvMF	**0.8531**	**0.7958**	**1.1337**
ConvMF+	**0.8549**	**0.7930**	**1.1279**
Improve	3.92%	2.79%	16.60%

Table 3: Overall test RMSE

ML-1m(Relatively dense dataset)

- PMF ≈ CTR ≈ CDL < ConvMF & ConvMF
- data가 상대적으로 dense할 땐 **shallow** document analysis는 **큰 도움이 되지 않았다**.
- 그럼에도 **Deeper Understanding**을 위시한 **ConvMF 방식은 가시적인 성능 개선**을 보였다.

ConvMF - Experiment

Model	Ratio of training set to the entire dataset (density)						
	20% (0.93%)	30% (1.39%)	40% (1.86%)	50% (2.32%)	60% (2.78%)	70% (3.25%)	80% (3.71%)
PMF	1.0168	0.9711	0.9497	0.9354	0.9197	0.9083	0.8971
CTR	1.0124	0.9685	0.9481	0.9337	0.9194	0.9089	0.8969
CDL	1.0044	0.9639	0.9377	0.9211	0.9068	0.8970	0.8879
ConvMF	**0.9745**	**0.9330**	**0.9063**	**0.8897**	**0.8726**	**0.8676**	**0.8531**
Improve	2.98%	3.20%	3.36%	3.41%	3.77%	3.27%	3.92%

Table 4: Test RMSE over various sparseness of training data on ML-1m dataset

Over Various density

- 매번 ConvMF가 우월한 성능을 보였다.
- Dataset density가 높아질수록 성능 격차를 더 벌리게 되었다.
- ConvMF의 CNN이 PMF와 잘 integrate되어 위와 같은 양상을 보이게 되었다.

ConvMF - Experiment

	Dataset		
Model	ML-1m	ML-10m	AIV
PMF	0.8971	0.8311	1.4118
CTR	0.8969	0.8275	1.5496
CDL	0.8879	0.8186	1.3594
ConvMF	**0.8531**	**0.7958**	**1.1337**
ConvMF+	**0.8549**	**0.7930**	**1.1279**
Improve	3.92%	2.79%	16.60%

Table 3: Overall test RMSE

AIV(Sparse dataset)

- CDL <<< ConvMF & ConvMF (16.6%)
- Sparse, skewed dataset에서는 현격한 성능개선을 이루어 냈다.
- 이는 ConvMF가 Contextual information을 고려한 document에 대한 deeper understanding 덕분이다.

ConvMF - Experiment

	Dataset		
Model	ML-1m	ML-10m	AIV
PMF	0.8971	0.8311	1.4118
CTR	0.8969	0.8275	1.5496
CDL	0.8879	0.8186	1.3594
ConvMF	**0.8531**	**0.7958**	**1.1337**
ConvMF+	**0.8549**	**0.7930**	**1.1279**
Improve	3.92%	2.79%	16.60%

Table 3: Overall test RMSE

Impact of Pre-trained word embedding(Sparse case)

- Data가 sparser해질수록 ConvMF+가 ConvMF보다 좋은 성능을 보였다.
- 이는 Pre-trained embedding의 syntactic, semantic info가 shortage of ratings를 보완하기 때문이다.

ConvMF - Experiment

Model	Dataset		
	ML-1m	ML-10m	AIV
PMF	0.8971	0.8311	1.4118
CTR	0.8969	0.8275	1.5496
CDL	0.8879	0.8186	1.3594
ConvMF	**0.8531**	**0.7958**	**1.1337**
ConvMF+	**0.8549**	**0.7930**	**1.1279**
Improve	3.92%	2.79%	16.60%

Table 3: Overall test RMSE

Impact of Pre-trained word embedding(Dense case)

- Dense할 땐 오히려 ConvMF가 더 좋은 성능을 내기도 하였다. (ML-1m)
- Dataset이 상대적으로 dense하면 Pre-trained word embedding 정보를 추가하기보단,
  Rating의 영향력을 더 중시해야한다. (Rating은 직접적으로 user x item 사이 관계를 반영하기 때문).

ConvMF - Experiment

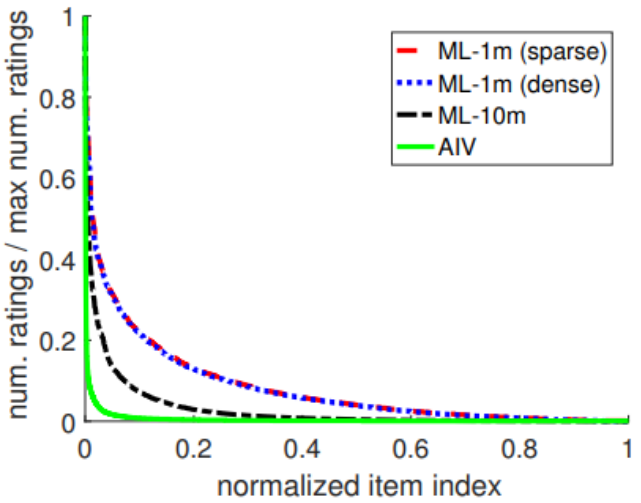


Figure 3: Skewness of the number of ratings for items on each dataset

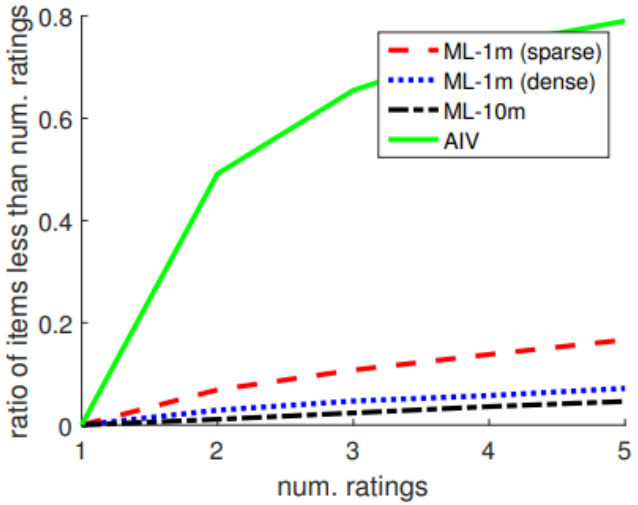


Figure 4: Ratio of items that have less than num. ratings (N) to each entire dataset

Skewness of Amazon dataset

- 극히 일부 item은 num. Ratings/ max num. Ratings ratio가 높고 나머지는 굉장히 낮다.
- 다른 dataset에 비해 num. Ratings에 따라 그 ratings이하의 items 비율 변화가 급격하다.
- 해당 dataset이 굉장히 skewness가 심하다는 것을 의미한다.

ConvMF - Experiment

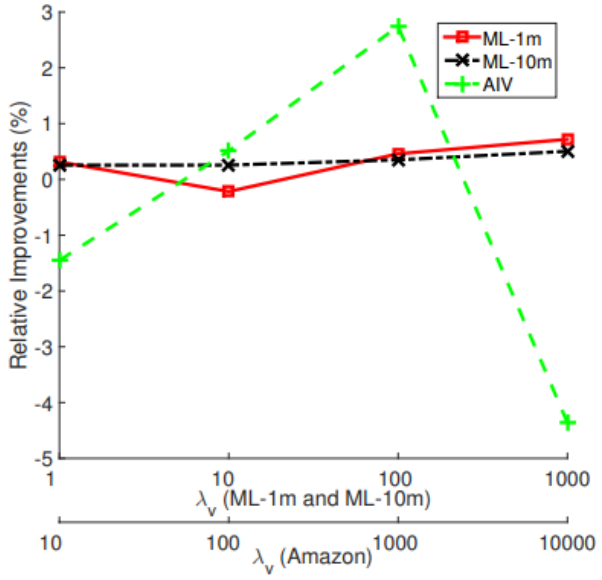


Figure 6: Relative improvements of ConvMF+ over ConvMF

Over various λ_V

- λ_V는 Rating과 document description사이 중요도를 balancing해준다.
- Dataset이 dense한 MovieLens의 경우, λ_V의 영향력이 미미했다.
- Sparse한 Amazon dataset의 경우, λ_V의 영향이 가시적으로 크게 나타났다.

ConvMF - Experiment

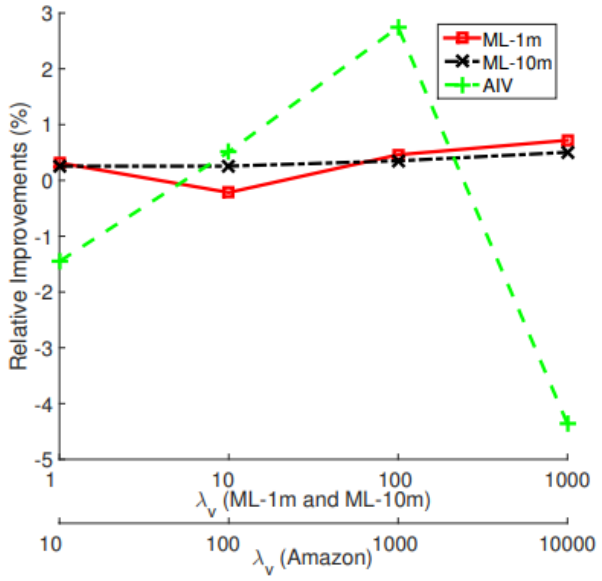


Figure 6: Relative improvements of ConvMF+ over ConvMF

Over various λ_V(Amazon Dataset)

- λ_V 10 to 1000 : ConvMF+의 상대적 성능이 증가한다.
- λ_V 1000 to 10000 : ConvMF+의 상대적 성능이 감소한다.

ConvMF - Experiment

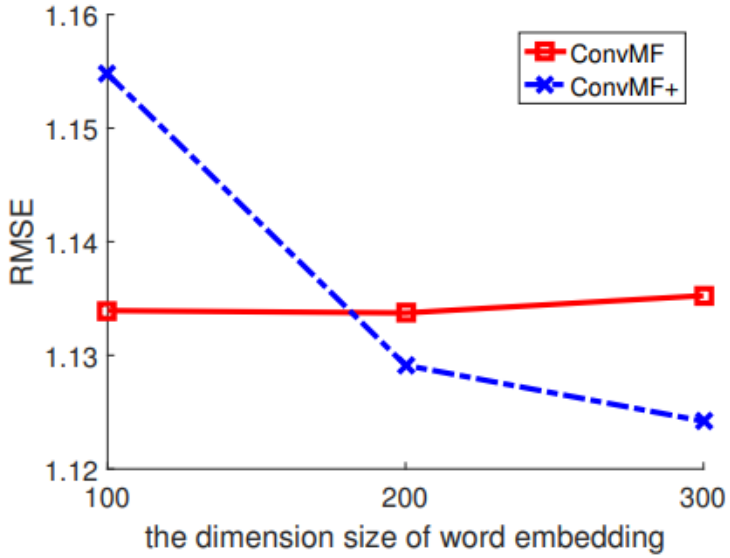


Figure 5: The effects of the dimension size of word embedding on Amazon dataset

Parameter analysis over Word vector dimension p

- p가 증가하자 ConvMF는 성능 저하, ConvMF+는 성능 개선이 일어났다. (Amazon dataset)
- Sparse할 때, p가 커짐에 따라 pre-trained word embedding model의 information이 풍부해진다.

ConvMF - Experiment

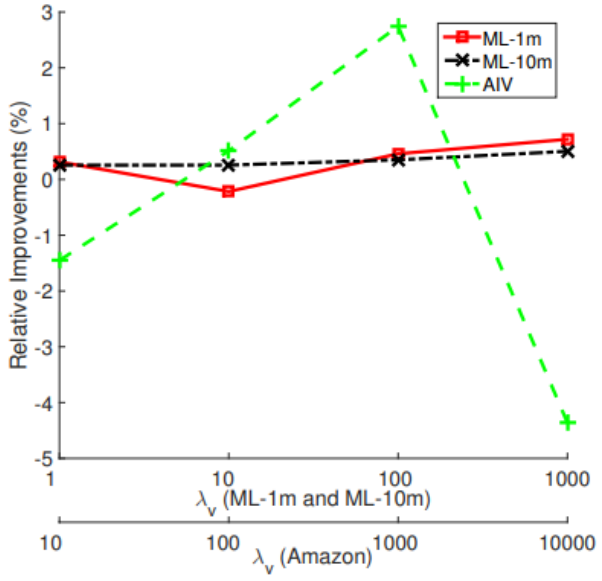
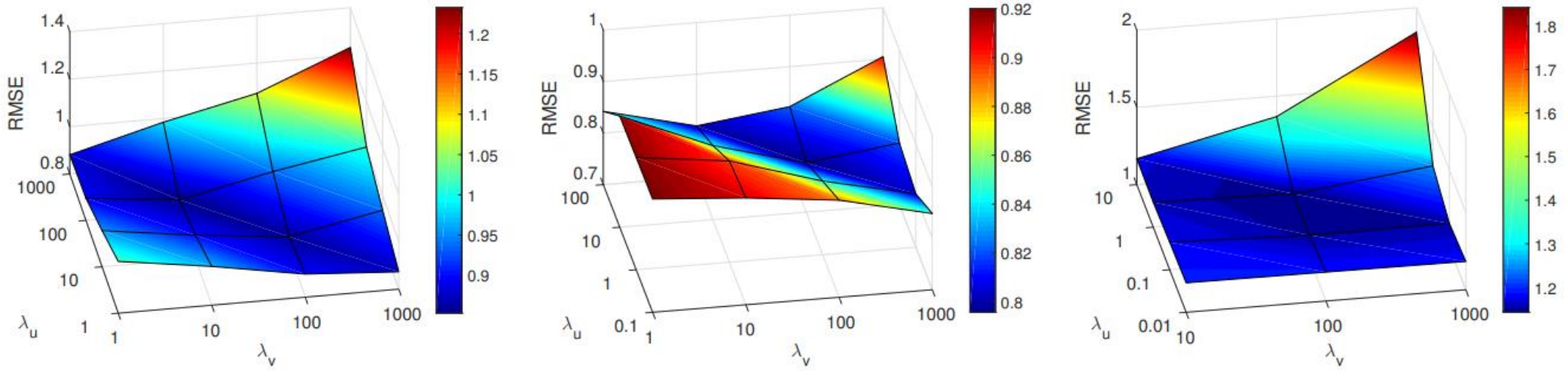


Figure 6: Relative improvements of ConvMF+ over ConvMF

Over various λ_V(Amazon Dataset)

- λ_V가 작을 때는 (+)가 상대적으로 Underfit되어있는 것이 λ_V가 증가하며 점차 해소되다가, 100 이후부터는 (+)가 오히려 Overfit하는 것이 이 양상의 원인이다.
- Amazon dataset은 sparse하기에 이 양상이 더 잘 드러난다.
- 그럼에도 적절한 λ_V라면 pre-trained word embedding을 활용한 ConvMF+가 성능을 더욱 높여주었다.

ConvMF - Experiment



(a) MovieLens-1m (b) MovieLens-10m (c) Amazon Instant Video

Parameter analysis over λ_V & λ_u

- Dataset이 sparse해질수록 λ_V는 증가하고, λ_u는 감소해야 좋은 결과를 산출하였다.
- λ_v 가 크다 > item latent model 변화 작다 > user latent model이 item latent model space에 project.
- Item latent model은 document description에 영향을 크게 받고 있다.
- 즉 sparse해질수록 document가 성능 개선에 영향을 주고 있고, ConvMF의 효과를 방증한다.
  (λ_v를 이용해 rating과 document 사이 balancing을 하면서)

ConvMF - Experiment

Phrase captured by W_c^{11}	max(c^{11})	Phrase captured by W_c^{86}	max(c^{86})
people **trust** the man	***0.0704***	betray his **trust** finally	***0.1009***
Test phrases for W_c^{11}	**c_{test}^{11}**	**Test phrases for W_c^{86}**	**c_{test}^{86}**
people **believe** the man	***0.0391***	betray his **believe** finally	*0.0682*
people **faith** the man	*0.0374*	betray his **faith** finally	***0.0693***
people **tomas** the man	*0.0054*	betray his **tomas** finally	*0.0480*

Table 5: Case study on two shared weights of ConvMF

Qualitative Evaluation(Subtle Contextual difference)

- 전자 : believe가 faith보다 높은 feature value 산출. (verb)
- 후자 : faith가 believe보다 높은 feature value 산출. (noun)
- 연관이 없는 tomas에 대해선 굉장히 낮은 feature value를 산출한다.
- 단순한 Contextual meaning 뿐 아니라 Subtle Contextual difference 역시 구별할 수 있다.

ConvMF - Conclusion

Impact of ConvMF

- ConvMF는 기존의 document desciription을 활용하였던 기법(CTR, CDL 등)들에서 더 나아가 CNN과 PMF를 접목시켜 Contextual information까지 포착해 Rating accuracy를 향상시키고자 하였다.
- 광범위한 실험을 통하여 여러 방면에서 ConvMF는 기존의 Competitors를 압도하는 성능 개선을 보였다.
- 특히나 rating이 sparse한 dataset에서 document에 대한 deeper understanding이 성능 개선에 유효한 영향을 끼치는 것을 확인하였다.
- 뿐만 아니라 더 나아가 미묘한 Contextual difference 역시 감지할 수 있음을 실험적으로 보였다.

AutoRec - Implementation

```python
class Data(Dataset):
    def __init__(self,train, base):
        super(Data, self).__init__()
        self.train = train
        self.base = base
        self.n_user, self.n_item = train.shape

    def __len__(self):
        if self.base == 'item':
            return self.n_item
        elif self.base == 'user':
            return self.n_user

    def __getitem__(self, idx):
        if self.base == 'item':
            return torch.tensor(self.train.iloc[:, idx].values).float()
        elif self.base == 'user':
            return torch.tensor(self.train.iloc[idx, :].values).float()
```

```python
class AutoRec(nn.Module):
    def __init__(self, input, hidden, output)
        super(AutoRec, self).__init__()
        self.enc = nn.Linear(input, hidden)
        self.dec = nn.Linear(hidden, output)
        self.activate = F.sigmoid


    def forward(self, x):
        x = self.activate(self.enc(x))
        x = self.dec(x)
        return x
```

AutoRec - Implementation

```
for x in trainloader:
    optimizer.zero_grad()
    x = x.to(config.device)
    mask = x > 0
    pred = model(x)
    loss = torch.mean(((x - pred)[mask])**2)
    loss.backward()
    optimizer.step()
    losses.append(np.sqrt(loss.item()))
```

```
EPOCH 15: TRAINING loss 0.9424899134346811
EPOCH 16: TRAINING loss 0.944043463195325
EPOCH 17: TRAINING loss 0.9501555757195482
EPOCH 18: TRAINING loss 0.9494641918918991
EPOCH 19: TRAINING loss 0.9698743369820528
EPOCH 20: TRAINING loss 0.9731743313445089
EPOCH 21: TRAINING loss 0.9612591393904667
EPOCH 22: TRAINING loss 0.9532326361770449
EPOCH 23: TRAINING loss 0.9463050780169887
EPOCH 24: TRAINING loss 0.9524248820920533
EPOCH 25: TRAINING loss 0.9252526664572827
EPOCH 26: TRAINING loss 0.9327948593011899
EPOCH 27: TRAINING loss 0.9211449407007336
EPOCH 28: TRAINING loss 0.9249190638146322
EPOCH 29: TRAINING loss 0.9120939904530396
EPOCH 30: TRAINING loss 0.9191502025580571
```

Thank you