HAI
—VIS

VisRec Tutorial
Session 4: CuratorNet

Denis Parra, Antonio Ossa-Guerra, **Manuel Cartagena**, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces

CuratorNet: Visually-aware Recommendation of Art Images

Pablo Messina*
Pontificia Universidad Católica
Santiago, Chile
pamessina@uc.cl

Manuel Cartagena
Pontificia Universidad Católica
Santiago, Chile
micartagena@uc.cl

Patricio Cerda
Pontificia Universidad Católica
Santiago, Chile
pcerdam@uc.cl

Felipe del Rio†
Pontificia Universidad Católica
Santiago, Chile
fidelrio@uc.cl

Denis Parra‡
Pontificia Universidad Católica
Santiago, Chile
dparra@ing.puc.cl

Context: Recommendation of Visual Art

Context: Recommendation of Visual Art

- Few Works in the area (compared to movies or music)

Context: Recommendation of Visual Art

- Few Works in the area (compared to movies or music)

- Most previous work focused on museums and cultural collections

Context: Recommendation of Visual Art

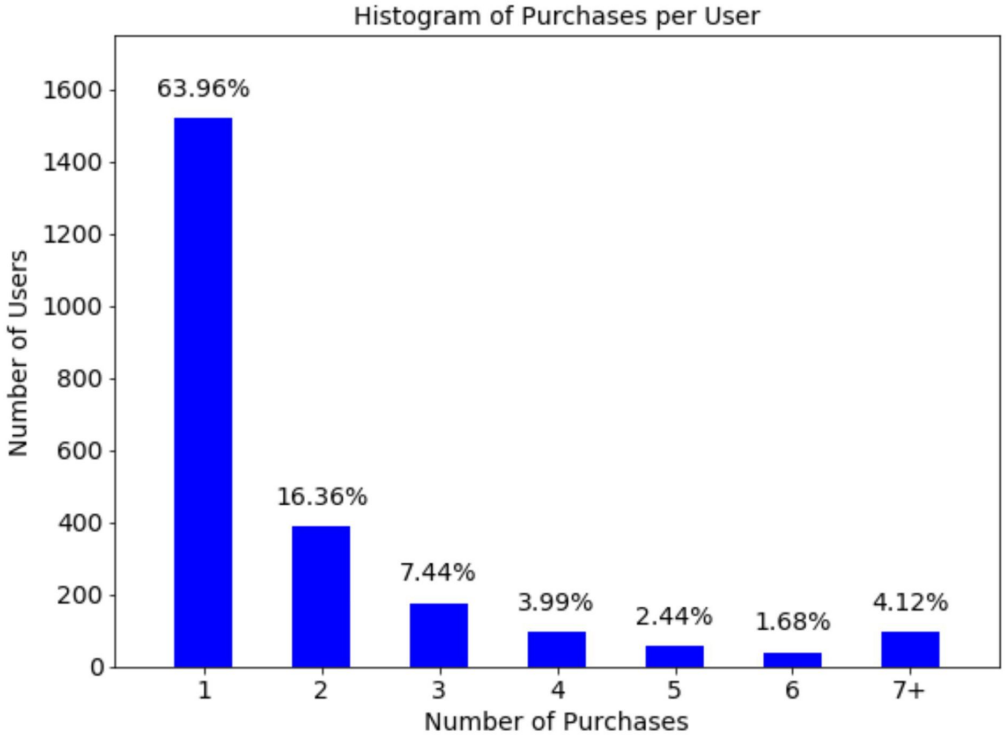
- Few Works in the area (compared to movies or music)

- Most previous work focused on museums and cultural collections

- Few studies about recommending paintings in a commercial setting.

Dataset: Ugallery

- 5,336 transactions (purchases)
- 2,378 users
- 6,040 paintings



Our Problem

- One-of-a-kind items: We have no explicit co-occurrence as in VBPR/VISTA (He & MacAuley, 2016)

Our Problem

- One-of-a-kind items: We have no explicit co-occurrence as in VBPR/VISTA (He & MacAuley, 2016)

- Small dataset, compared to YouTube as well as compared to VBPR/VISTA

Our Problem

- One-of-a-kind items: We have no explicit co-occurrence as in VBPR/VISTA (He & MacAuley, 2016)

- Small dataset, compared to YouTube as well as compared to VBPR/VISTA

- Ideas to addressing our problem: YouTube does not learn a explicit user latent vector and VBPR performs learning by sampling negative feedback.

IDEA: ¿What about combining ***visual content*** *with* ***collaborative information without the need of explicit user latent factors****?*

CuratorNet

A neural network architecture for visually-aware recommendation of art images.

Original implementation:
https://github.com/ialab-puc/CuratorNet

Model

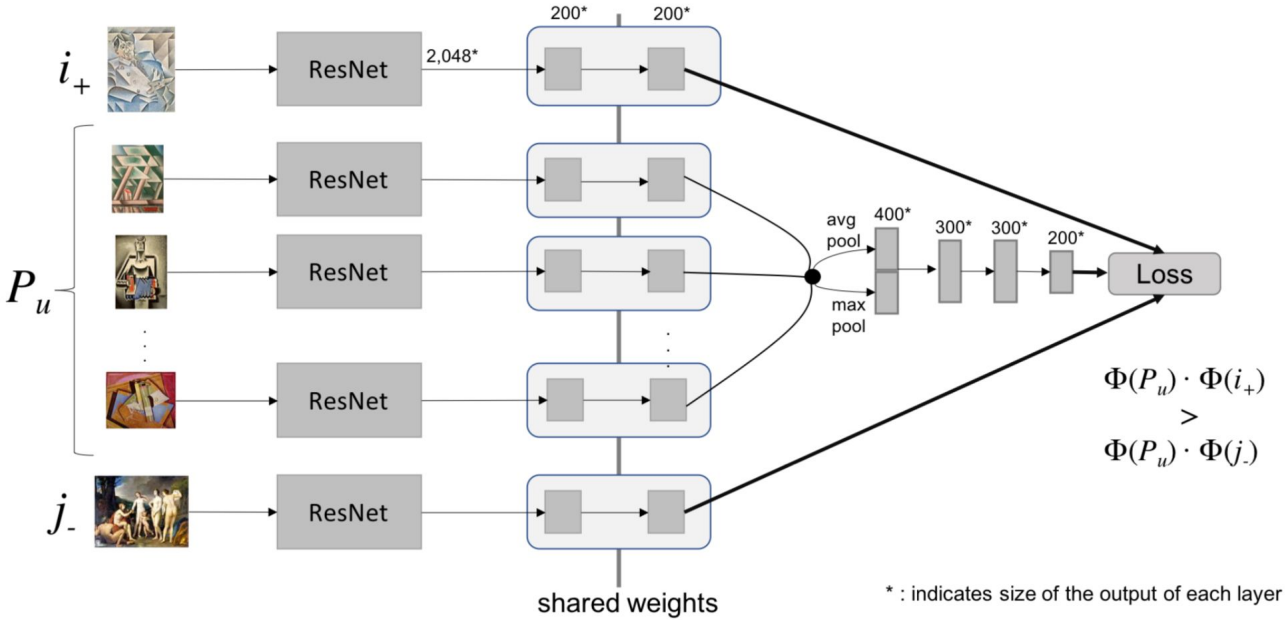


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

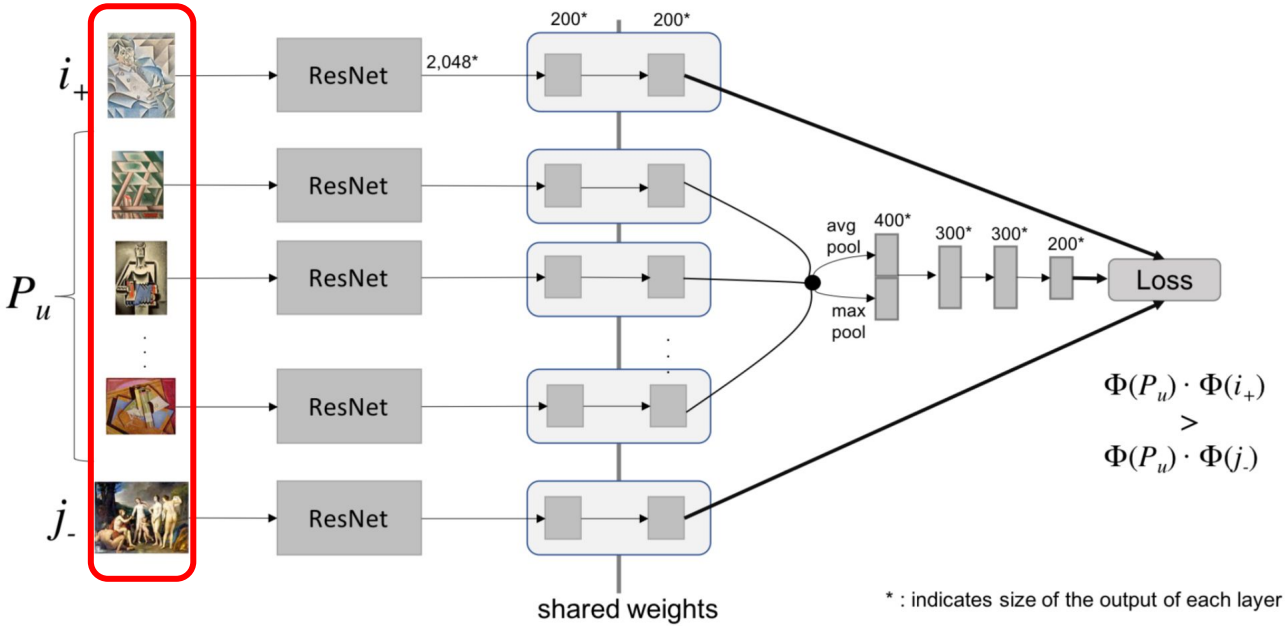


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

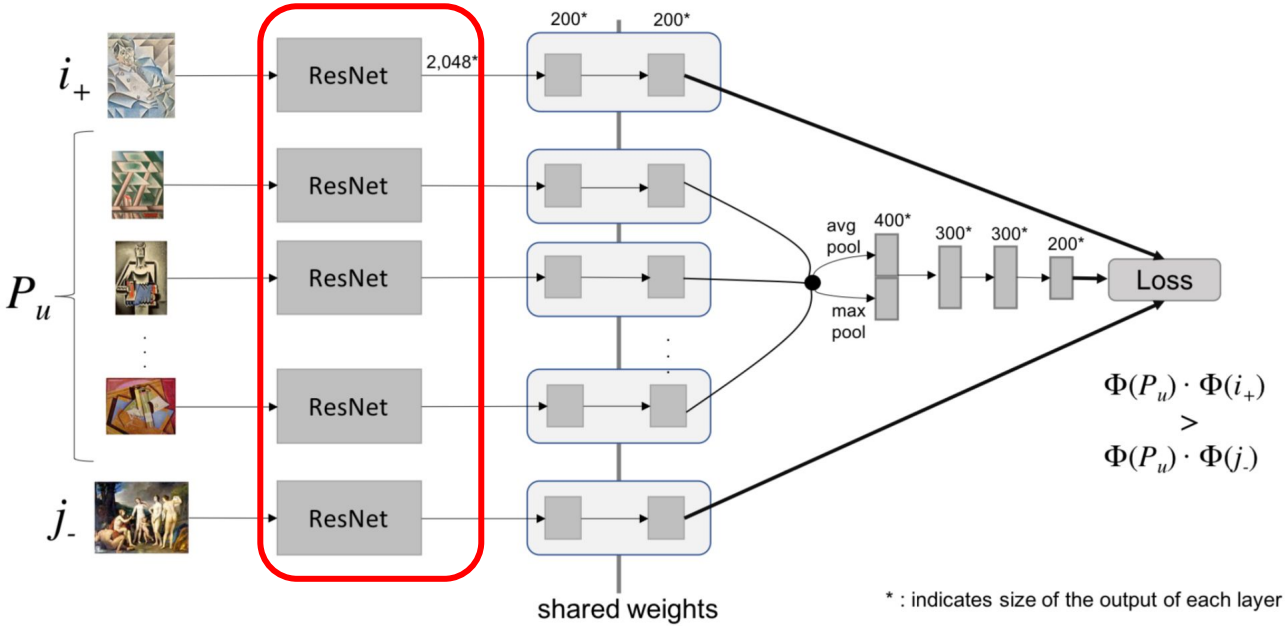


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

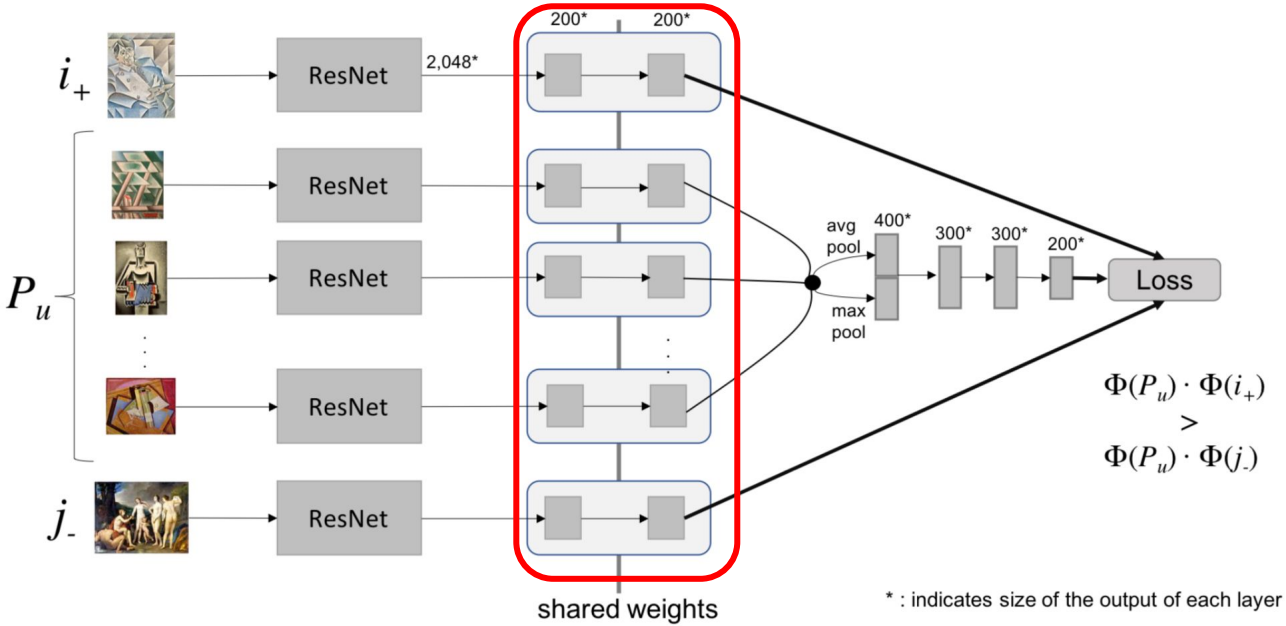


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

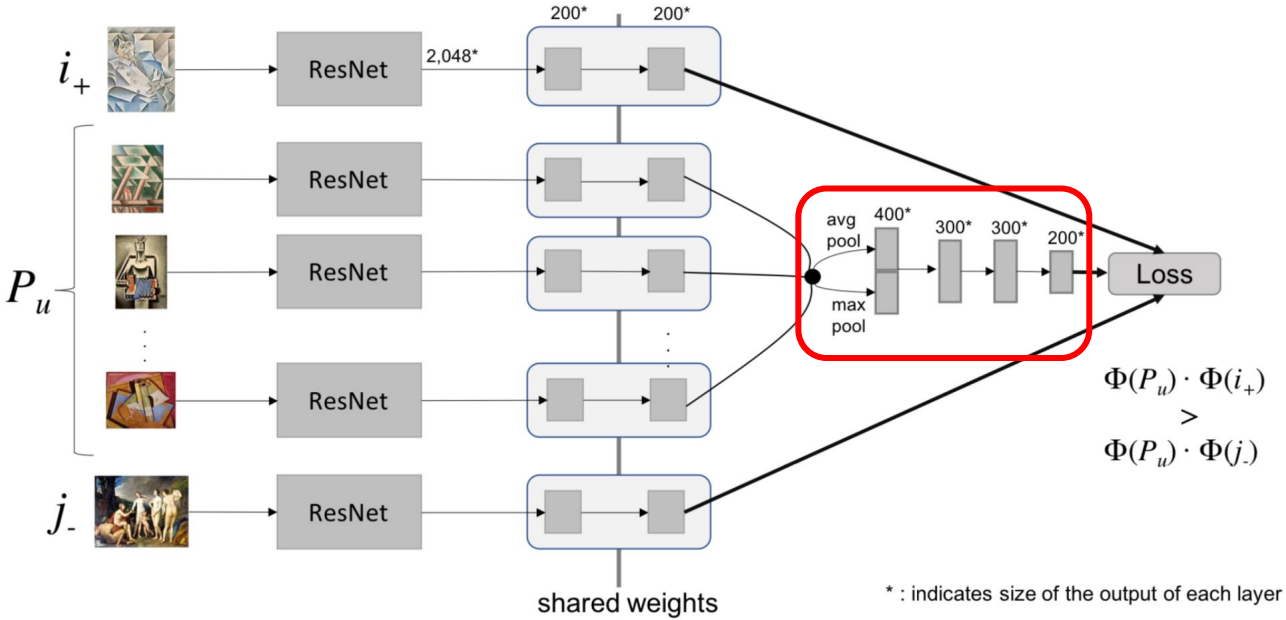


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

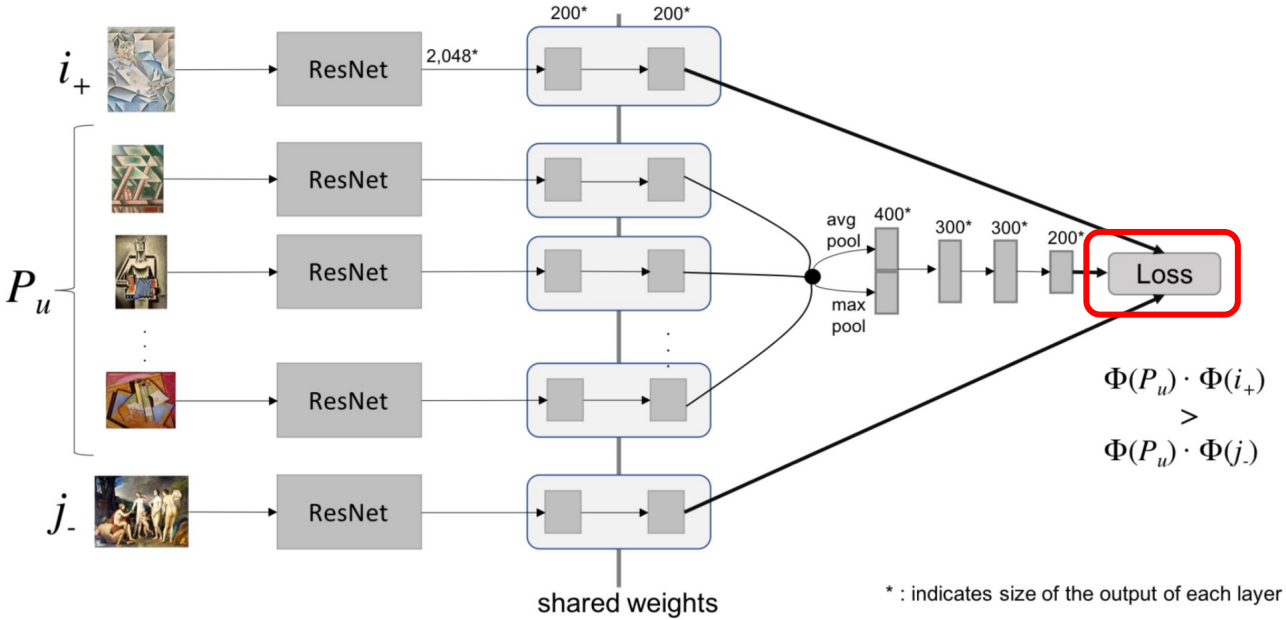


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model

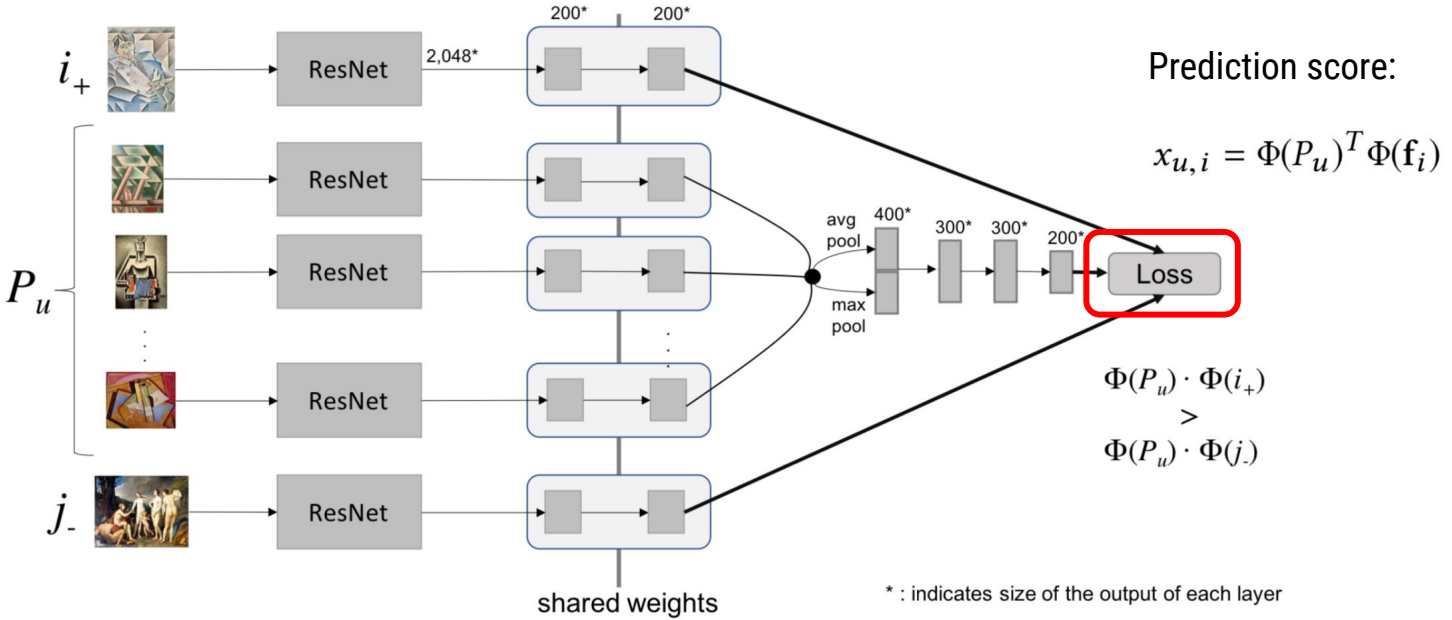


Figure 2: Architecture of CuratorNet showing in detail the layers with shared weights for training.

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py / Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) | 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        profile = torch.cat(
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        profile = torch.cat(
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py / Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) | 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        profile = torch.cat(
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py / Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        p[illegible]
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py / Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) | 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        profile = torch.cat(
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Model Code

main VisualRecSys-Tutorial-IUI2021 / models / curatornet.py / Jump to

mcartagenah add curatornet model training notebook

1 contributor

151 lines (122 sloc) | 5.1 KB

```python
"""CuratorNet implementation in PyTorch
"""
import torch
import torch.nn as nn
import torch.nn.functional as F


class CuratorNet(nn.Module):
    """CuratorNet model architecture from 'CuratorNet: A Neural
    Network for Visually-aware Recommendation of Art Images'.
    """

    def __init__(self, embedding, input_size=2048):
        super().__init__()

        # Embedding
        self.embedding = nn.Embedding.from_pretrained(embedding, freeze=True)

        # Common section
        self.selu_common1 = nn.Linear(input_size, 200)
        self.selu_common2 = nn.Linear(200, 200)

        # Profile section
        self.maxpool = nn.AdaptiveMaxPool2d((1, 200))
        self.avgpool = nn.AdaptiveAvgPool2d((1, 200))
        self.selu_pu1 = nn.Linear(200 + 200, 300)
        self.selu_pu2 = nn.Linear(300, 300)
        self.selu_pu3 = nn.Linear(300, 200)

        # Random weight initialization
        self.reset_parameters()
```

```python
    def forward(self, profile, pi, ni):
        """Forward pass of the model.

        Feed forward a given input (batch). Each object is expected
        to be a Tensor.

        Args:
            profile: User profile items embeddings, as a Tensor.
            pi: Positive item embedding, as a Tensor.
            ni: Negative item embedding, as a Tensor.

        Returns:
            Network output (scalar) for each input.
        """
        # Load embedding data
        profile = self.embedding(profile)
        pi = self.embedding(pi)
        ni = self.embedding(ni)

        # Positive item
        pi = F.selu(self.selu_common1(pi))
        pi = F.selu(self.selu_common2(pi))

        # Negative item
        ni = F.selu(self.selu_common1(ni))
        ni = F.selu(self.selu_common2(ni))

        # User profile
        profile = F.selu(self.selu_common1(profile))
        profile = F.selu(self.selu_common2(profile))
        profile = torch.cat(
            (self.maxpool(profile), self.avgpool(profile)), dim=-1
        )
        profile = F.selu(self.selu_pu1(profile))
        profile = F.selu(self.selu_pu2(profile))
        profile = F.selu(self.selu_pu3(profile))

        # x_ui > x_uj
        x_ui = torch.bmm(profile, pi.unsqueeze(-1))
        x_uj = torch.bmm(profile, ni.unsqueeze(-1))

        return x_ui - x_uj
```

Loss function: Sigmoid Cross-Entropy Loss

$$\mathcal{L} = -\sum_{\mathcal{D}_S} c \ln(\sigma(x_{u,i,j})) + (1-c)\ln(1-\sigma(x_{u,i,j})) + \lambda_\Theta ||\Theta||^2$$

Loss function: Sigmoid Cross-Entropy Loss

Class:
0 = wrongly ranked
1 = correctly ranked

Probability that user *u* prefers *i* over *j*

Probability that user *u doesn't* prefers *i* over *j*

$$\mathcal{L} = -\sum_{\mathcal{D}_S} c \ln(\sigma(x_{u,i,j})) + (1 - c) \ln(1 - \sigma(x_{u,i,j})) + \lambda_\Theta ||\Theta||^2$$

L2 Regularization hyperparameter

Sigmoid function ($\boldsymbol{\sigma}$)

$$P(i >_u j|\Theta) = \sigma(x_{u,i,j}) = \frac{1}{1 + e^{-(x_{u,i} - x_{u,j})}}$$

Loss function Code

main VisualRecSys-Tutorial-IUI2021 / 3 - (CuratorNet) Training procedure.ipynb

L2 regularization

Sigmoid cross entropy

```
# Training setup
print("\nSetting up training")
optimizer = optim.Adam(
    model.parameters(),
    lr=SETTINGS["optimizer:lr"],
    weight_decay=SETTINGS["optimizer:weight_decay"],
)
criterion = nn.BCEWithLogitsLoss(reduction="sum")
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode="max", factor=SETTINGS["scheduler:factor"],
    patience=SETTINGS["scheduler:patience"], verbose=True,
    threshold=SETTINGS["scheduler:threshold"],
)
```

Training the model

- Similar to BPR: Given a training set D_S of triples (p, i, j) we aim that our model score like:

$$\vec{u_p} \cdot \vec{i} > \vec{u_p} \cdot \vec{j}$$

- Unlike BPR, we do not randomly sample negative examples for the training set D_S

Sampling guidelines for triples

- Based on findings of our previous work (favorite artist)
- Use notion of visual clusters:

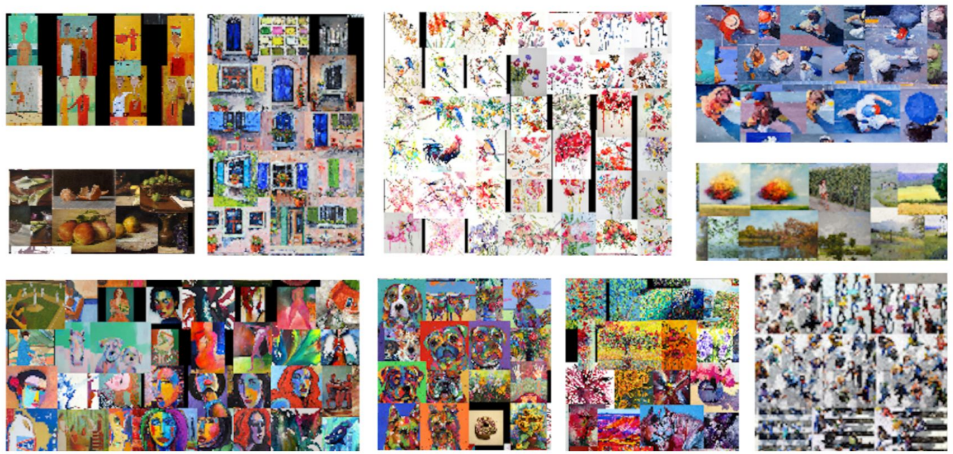


Figure 3: Examples of visual clusters automatically generated to sample triples for the training set.

Effect of sampling guidelines

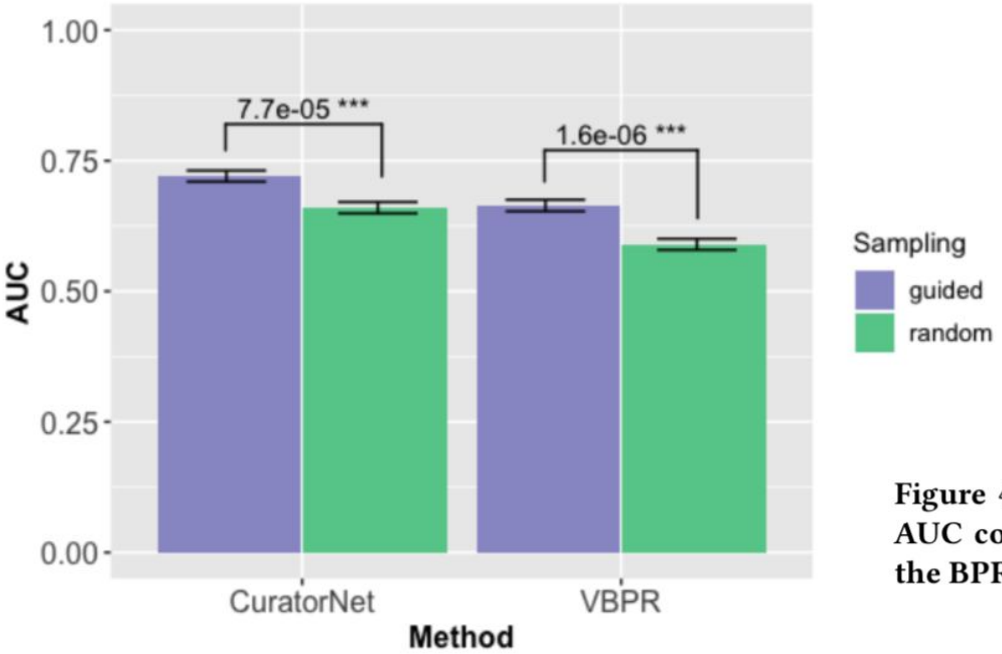
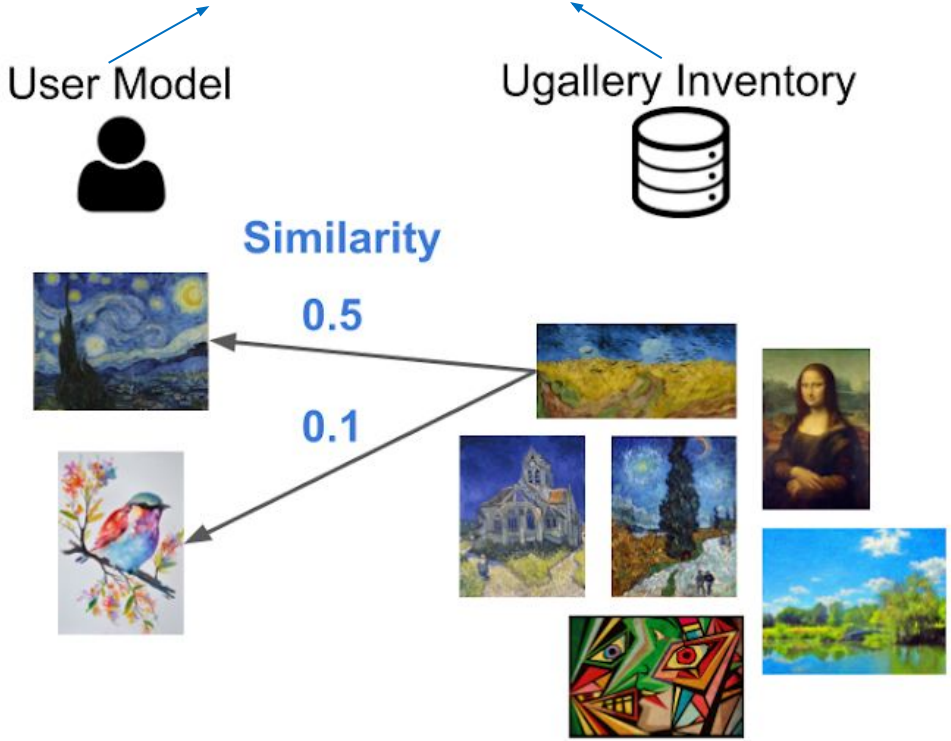


Figure 4: The sampling guidelines had a positive effect on AUC compared to random negative sampling for building the BPR training set.

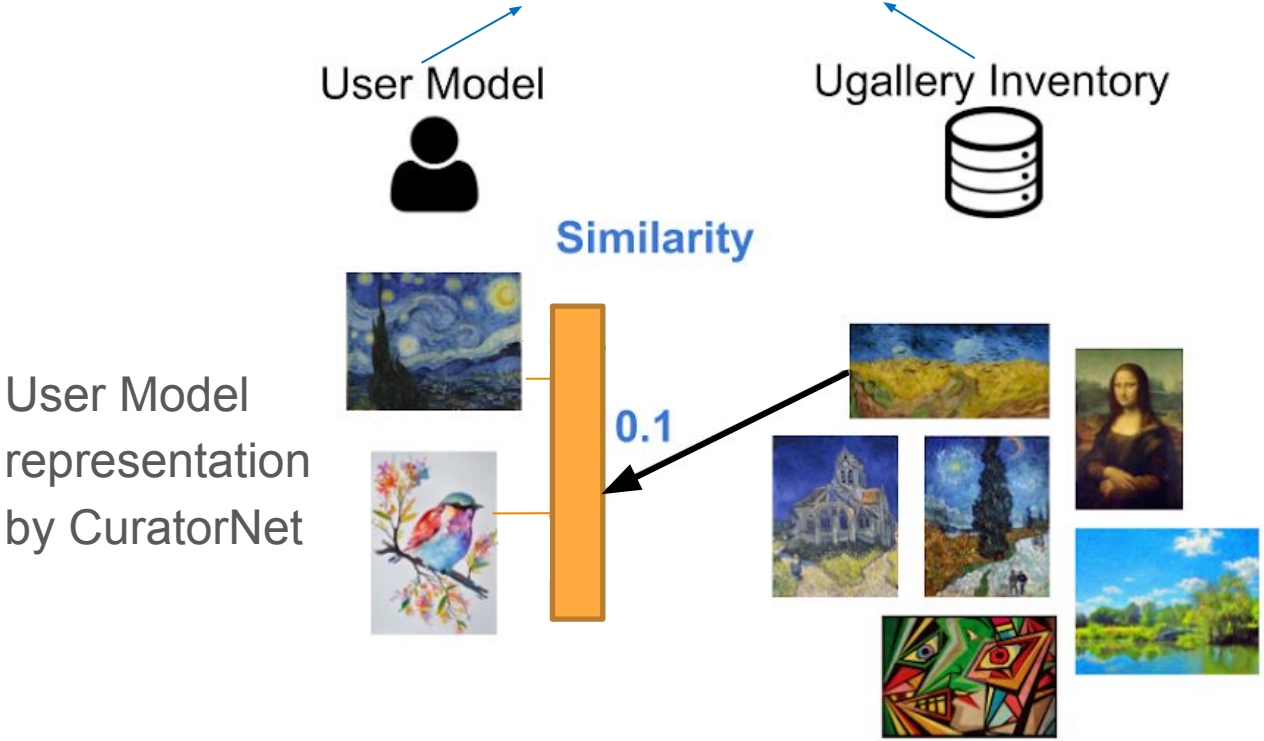
Content-based recommendation: VisRank (baseline)

$$s(u,i) = score(\ ContentBasedProfile(u),\ Content(i)\ )$$



Content-based recommendation: CuratorNet

$$s(u,i) = score(\ ContentBasedProfile(u),\ Content(i)\ )$$



Results Wikimedia (Random sampling)

AUC	RR	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
.66931	.01955	.03803	.00190	.02226	.07884	.00078	.02943

Results Ugallery (with guidelines)

AUC	R@20	P@20	nDCG@20	R@100	P@100	nDCG@100
.7204	.1683	.0106	.0966	.2399	.0030	.0923




VisRec: A Hands-on Tutorial on Deep Learning for Visual Recommender Systems

Denis Parra, Antonio Ossa-Guerra, **Manuel Cartagena**, *Patricio Cerda-Mardini, Felipe del Río
Pontificia Universidad Católica de Chile
*MindsDB

26th ACM Conference on Intelligent User Interfaces

References

Messina, P., Cartagena, M., Cerda, P., del Rio, F., & Parra, D. (2020). CuratorNet: Visually-aware Recommendation of Art Images. arXiv preprint arXiv:2009.04426.

He, R., & McAuley, J. (2016, February). VBPR: visual bayesian personalized ranking from implicit feedback. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 30, No. 1).

Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems (pp. 191-198).

Messina, P., Dominguez, V., Parra, D., Trattner, C., & Soto, A. (2019). Content-based artwork recommendation: integrating painting metadata with neural and manually-engineered visual features. User Modeling and User-Adapted Interaction, 29(2), 251-290.

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2012). BPR: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618.

Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A Visually, Socially, and Temporally-aware Model for Artistic Recommendation. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16).

Hands-On

Inspiration 1: VBPR

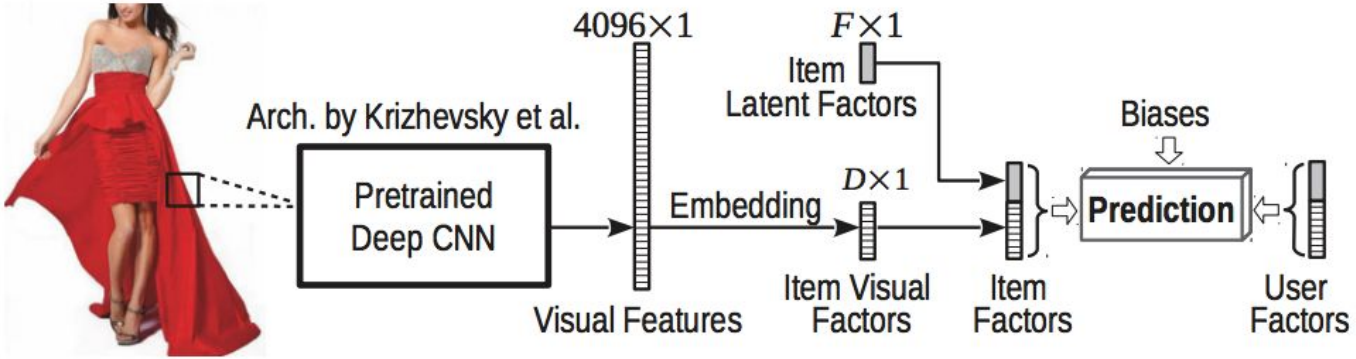
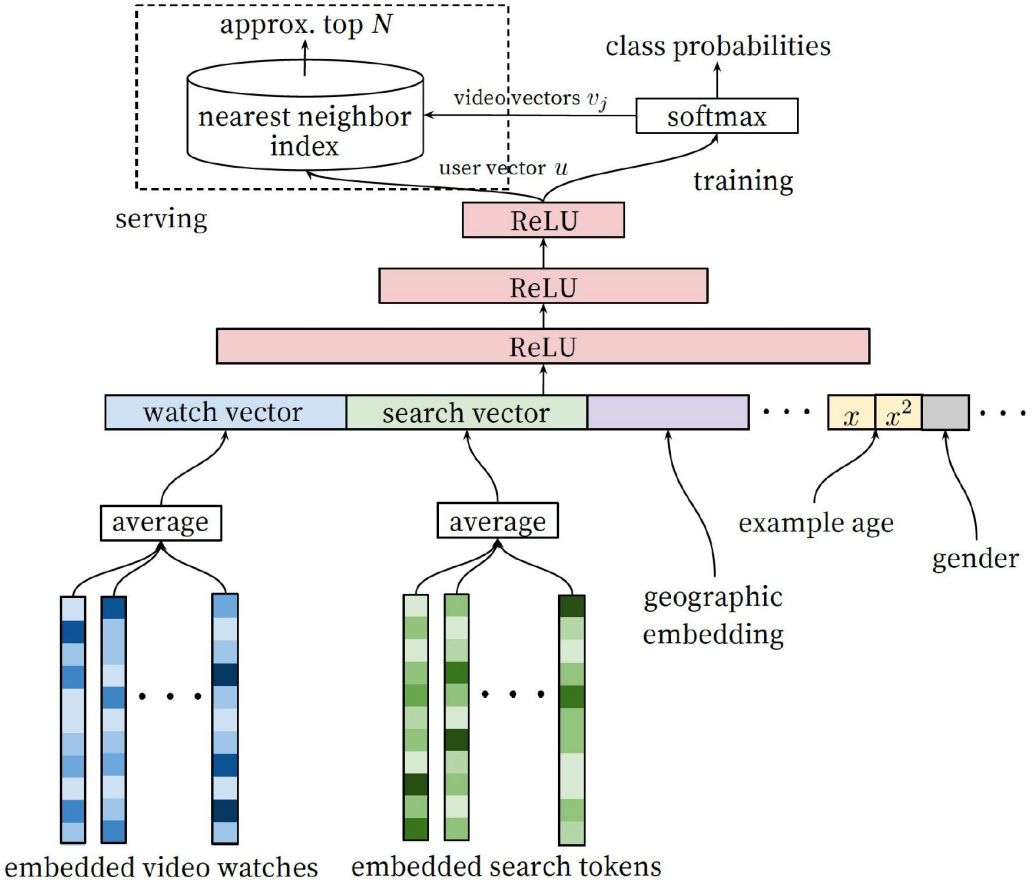


Figure 1: Diagram of our preference predictor. Rating dimensions consist of visual factors and latent (non-visual) factors. Inner products between users and item factors model the compatibility between users and items.

Inspiration 2: Youtube

- Deep Neural Networks for YouTube Recommendations (Covington et.al, 2016)



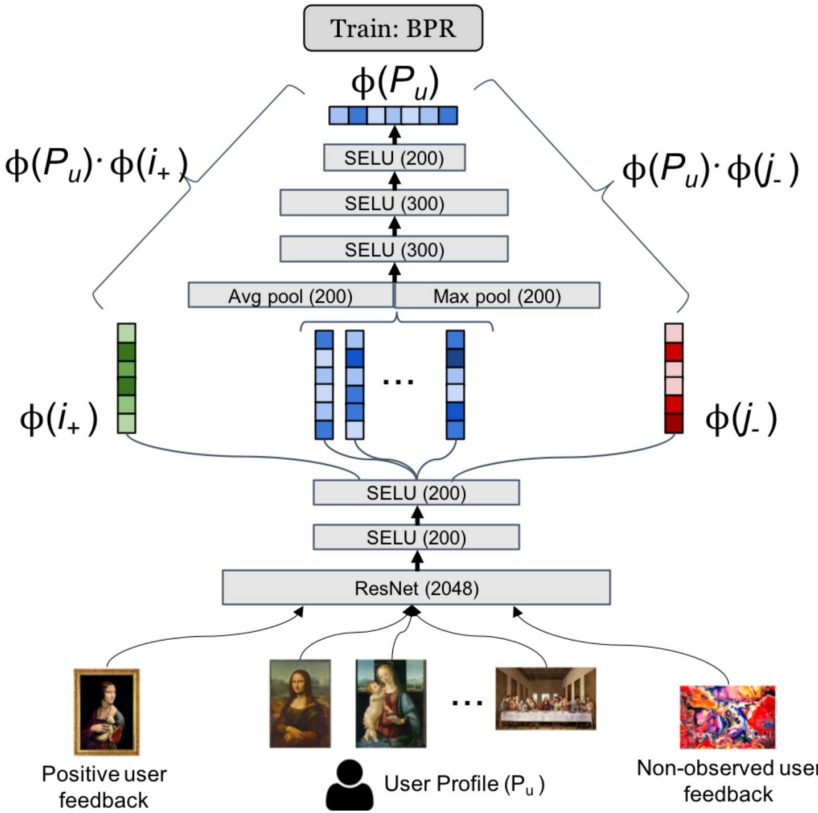


Figure 1: General architecture of CuratorNet. Parameters are learned via BPR [34]. The first two SELU layers have shared weights, similar to triplet loss models [39, 45].

The guidelines

(1) Removing item from purchase basket, and predicting this missing item.

(2) Sort items purchased sequentially, and then predict next purchase in basket.

(3) Recommending visually similar artworks from the favorite artists of a user.

(4) Recommending profile items from the same user profile.

(5) Create an artificial user profile of a single item purchased, and recommending profile items given this artificially created user profile.

(6) Create artificial profile with a single item, then recommend visually similar items from the same artist.

Offline Evaluation (Purchase Records)