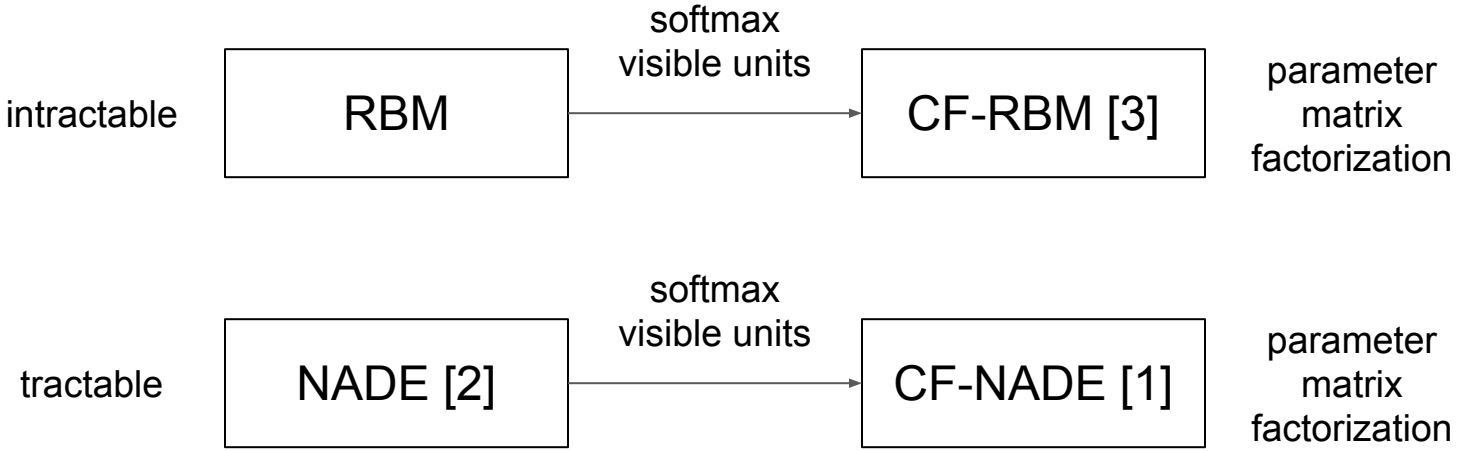
NADE, GAN Recommender

유찬우

Taxonomy of Generative Models

- Explicit Density
  - Tractable Density
    - NADE
    - Pixel RNN/CNN
  - Approximate Density
    - Variational Autoencoder
    - RBM
- Implicit Density
  - GAN

A Neural Autoregressive Approach to Collaborative Filtering [1]


intractable
RBM
softmax
visible units
CF-RBM [3]
parameter
matrix
factorization
tractable
NADE [2]
softmax
visible units
CF-NADE [1]
parameter
matrix
factorization

RBM

$$\frac{\partial - \log p(\mathbf{x}^{(t)})}{\partial \theta} = \underbrace{\mathrm{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \theta} \middle| \mathbf{x}^{(t)}\right]}_{\text{positive phase}} - \underbrace{\mathrm{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}\right]}_{\text{negative phase}}$$

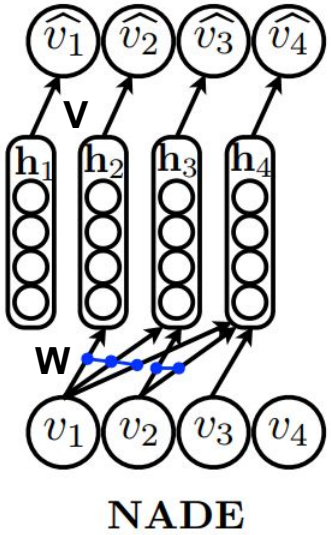
=> "negative phase" is intractable

=> Contrastive Divergence

NADE

$$p(\mathbf{v}) = \prod_{i=1}^{D} p(v_i | \mathbf{v}_{<i})$$

$$p(v_i = 1 | \mathbf{v}_{<i}) = \mathrm{sigm}\left(b_i + \mathbf{V}_{i,:}\mathbf{h}_i(\mathbf{v}_{<i})\right)$$



NADE

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathrm{sigm}\left(\mathbf{c} + \mathbf{W}_{:,<i}\mathbf{v}_{<i}\right)$$

loss = - log p(**v**)

NADE can be trained by SGD

RBM

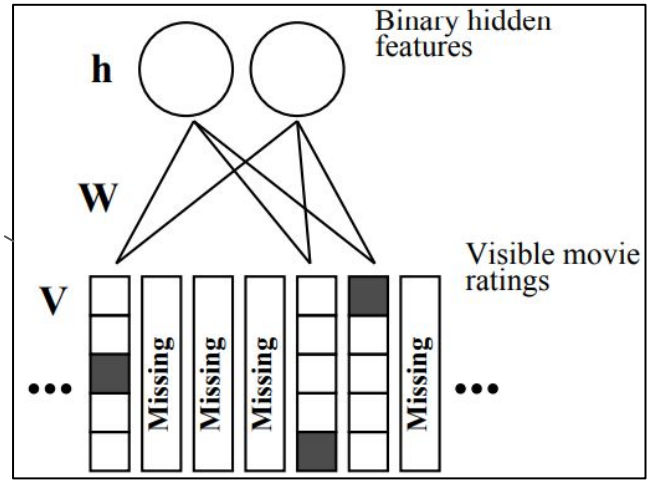
softmax visible units

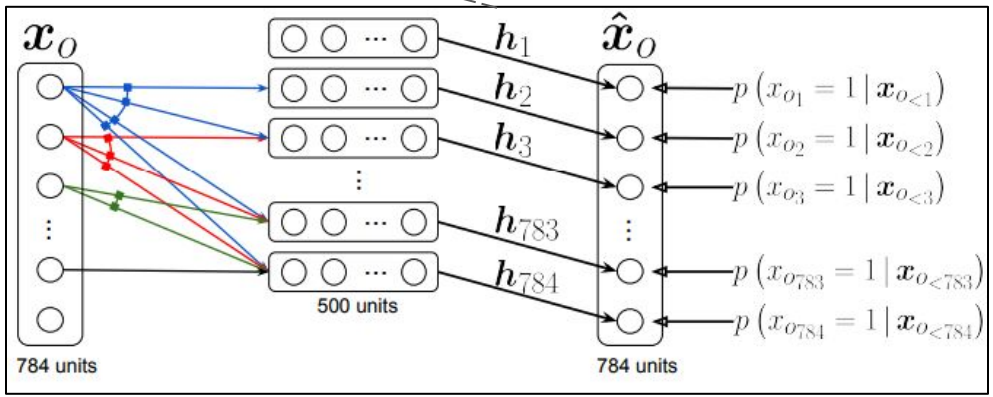
CF-RBM [3]

NADE [2]

softmax visible units

CF-NADE [1]





$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)}$$

$$p(h_j = 1|\mathbf{V}) = \sigma\left(b_j + \sum_{i=1}^{m} \sum_{k=1}^{K} v_i^k W_{ij}^k\right)$$

NADE

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \text{sigm}\left(\mathbf{c} + \mathbf{W}_{:,<i}\mathbf{v}_{<i}\right)$$

$$p(v_i = 1|\mathbf{v}_{<i}) = \text{sigm}\left(b_i + \mathbf{V}_{i,:}\mathbf{h}_i(\mathbf{v}_{<i})\right)$$

CF-RBM

$$p(h_j = 1|\mathbf{V}) = \sigma\left(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k\right)$$

$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K}\exp\left(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)}$$

CF-NADE

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \tanh\left(\mathbf{c} + \sum_k \mathbf{W}_{:,<i}^k \mathbf{v}_{<i}^k\right)$$

$$\mathbf{h}\left(\mathbf{r}_{m_{o<i}}\right) = \mathbf{g}\left(\mathbf{c} + \sum_{j<i} \mathbf{W}_{:,m_{oj}}^{r_{m_{oj}}}\right)$$

$$p\left(r_{m_{oi}} = k|\mathbf{r}_{m_{o<i}}\right) = \frac{\exp\left(s_{m_{oi}}^k\left(\mathbf{r}_{m_{o<i}}\right)\right)}{\sum_{k'=1}^{K}\exp\left(s_{m_{oi}}^{k'}\left(\mathbf{r}_{m_{o<i}}\right)\right)}$$

$$s_{m_{oi}}^k\left(\mathbf{r}_{m_{o<i}}\right) = b_{m_{oi}}^k + \mathbf{V}_{m_{oi},:}^k \mathbf{h}\left(\mathbf{r}_{m_{o<i}}\right)$$

Question

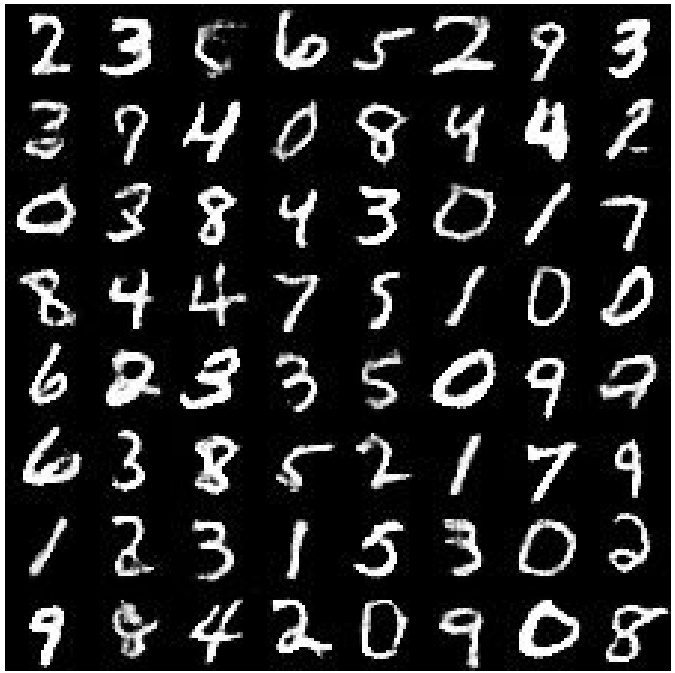
- order를 (v1, v2, v3, v4, v5)와 같이 잡았을 때, user가 rating을 준 영화가 (v1, v5)라면 training과 이 user에 대한 prediction(v3, v4에 대한)이 어떻게 이루어지는지?
- *"As in NADE, the ordering o in CF-NADE must be predefined and fixed during training for each user." [1]*
- *"The fixed ordering of the variables in a NADE model makes the exact calculation of arbitrary conditional probabilities computationally intractable." [4]*
- *"for any inference task would be to train as many NADE models as there are possible orderings of the input variables. Obviously, this approach, requiring O(D!) time and memory" [5]*

Implementation

CF-NADE

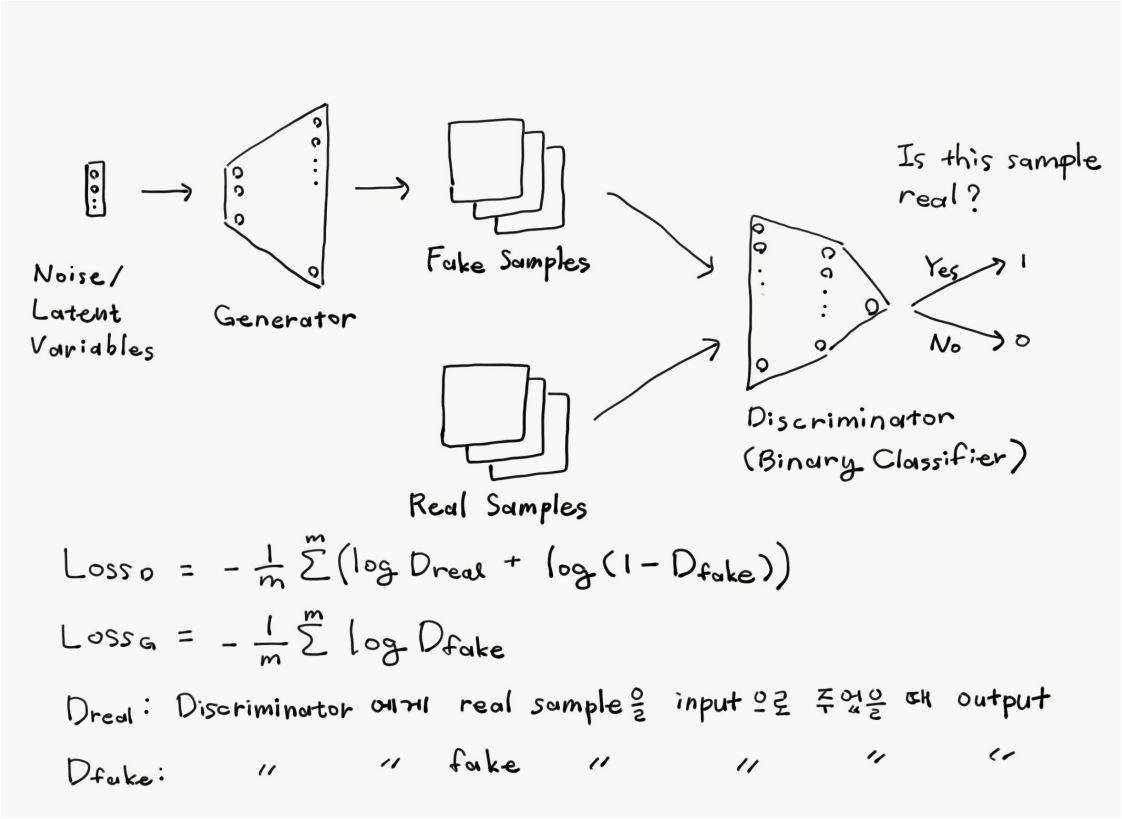
IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models [6]

Examples generated by GAN



source: https://github.com/carpedm20/DCGAN-tensorflow

GAN (Generative Adversarial Networks)



$$\text{Loss}_D = -\frac{1}{m}\sum^{m}(\log D_{real} + \log(1 - D_{fake}))$$

$$\text{Loss}_G = -\frac{1}{m}\sum^{m}\log D_{fake}$$

D_{real} : Discriminator 에게 real sample 을 input 으로 주었을 때 output

D_{fake} : 〃 〃 fake 〃 〃 〃 〃

Application: Item Recommendation

- Datasets: Movielens (100k) and Netflix

- *"we have chosen a basic matrix factorization model to implement"*
  - Generator: Matrix Factorization Model
  - $r_{u,i} = b_i + \mathbf{v}_u^T \mathbf{v}_i$

- *"we regard the 5-star ratings in both Movielens and Netflix as positive feedback and treat all other entries as unknown feedback, because we mainly focus on the implicit feedbacks problem."*
  - 5-star rating => 1.0
  - all other entries => unobserved

Implementation

NEWS Recommendation Using Generative adversarial neural network

- Discriminator
  - Matrix Factorization
  - $y_i = sigm(\mathbf{v}_u^T \mathbf{v}_i + b_i)$
  - Data
    - (user, item, label)
    - label: real sample이면 1, fake이면 0
- Generator
  - Matrix Factorization
  - Fake Sample: user u가 읽었을 것이라고 예상되는 item 리스트
  - Sampling 방법
    - user u에 대한 모든 item들의 rating 값(= $\mathbf{v}_u^T \mathbf{v}_i + b_i$)에 softmax를 적용해서 item별로 샘플링될 확률을 계산한 뒤, 일정 개수를 샘플링

Question 1

- Discriminator가 잘 훈련된 matrix factorization model이 될 경우에, Discriminator로부터 평가받는 또 다른 matrix factorization model을 생성하는 것이 의미가 있는지?

Question 2

- *"we regard the 5-star ratings in both Movielens and Netflix as positive feedback and treat all other entries as unknown feedback, because we mainly focus on the implicit feedbacks problem."*

- rating이 한 종류밖에 없는 경우, Matrix Factorization이 의미가 있는지?
  - $r_{u,i} = b_i + \mathbf{v}_u^T \mathbf{v}_i$
  - unobserved rating을 0으로 놓지 않고, observed rating에 대해서만 loss를 계산해서 training
  - observed rating이 1.0 한 종류밖에 없다면
  - $\mathbf{v}_u$, $\mathbf{v}_i$가 100-dimensional vector일 때, 모든 component의 값을 0.1로 놓고 $b_i = 0$으로 놓으면, training loss는 0이 된다.

NADE

We start with the description of the original NADE. NADE is a generative model over vectors of binary observations $\mathbf{v} \in \{0,1\}^D$. Through the probability chain rule, it decomposes $p(\mathbf{v}) = \prod_{i=1}^{D} p(v_i|\mathbf{v}_{<i})$ and computes all $p(v_i|\mathbf{v}_{<i})$ using the feed-forward architecture

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \text{sigm}\left(\mathbf{c} + \mathbf{W}_{:,<i}\mathbf{v}_{<i}\right), \qquad p(v_i = 1|\mathbf{v}_{<i}) = \text{sigm}\left(b_i + \mathbf{V}_{i,:}\mathbf{h}_i(\mathbf{v}_{<i})\right) \tag{1}$$

for $i \in \{1, \ldots, D\}$, where $\text{sigm}(x) = 1/(1+\exp(-x))$, $\mathbf{W} \in \mathbb{R}^{H \times D}$ and $\mathbf{V} \in \mathbb{R}^{D \times H}$ are connection parameter matrices, $\mathbf{b} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^H$ are bias parameter vectors, $\mathbf{v}_{<i}$ is the subvector $[v_1, \ldots, v_{i-1}]^\top$ and $\mathbf{W}_{:,<i}$ is a matrix made of the $i-1$ first columns of $\mathbf{W}$.

Discriminator

```
user_pos_train[uid].append(iid)

generate_for_d(sess, generator, DIS_TRAIN_FILE)

def generate_for_d(sess, model, filename):
    data = []
    for u in user_pos_train:
        pos = user_pos_train[u]

        rating = sess.run(model.all_rating, {model.u: [u]})
        rating = np.array(rating[0]) / 0.2  # Temperature
        exp_rating = np.exp(rating)
        prob = exp_rating / np.sum(exp_rating)

        neg = np.random.choice(np.arange(ITEM_NUM), size=len(pos), p=prob)
        for i in range(len(pos)):
            data.append(str(u) + '\t' + str(pos[i]) + '\t' + str(neg[i]))

self.all_rating = tf.matmul(self.u_embedding, self.item_embeddings, transpose_a=False,
                            transpose_b=True) + self.item_bias
```

Discriminator

```
input_user, input_item, input_label = ut.get_batch_data(DIS_TRAIN_FILE, index, BATCH_SIZE)
```

```
def get_batch_data(file, index, size):  #
    user = []
    item = []
    label = []
    for i in range(index, index + size):
        line = linecache.getline(file, i)
        line = line.strip()
        line = line.split()
        user.append(int(line[0]))
        user.append(int(line[0]))
        item.append(int(line[1]))
        item.append(int(line[2]))
        label.append(1.)
        label.append(0.)
    return user, item, label
```

```
sess.run(discriminator.d_updates,
         feed_dict={discriminator.u: input_user, discriminator.i: input_item,
                    discriminator.label: input_label})
```

Discriminator

```
self.u_embedding = tf.nn.embedding_lookup(self.user_embeddings, self.u)
self.i_embedding = tf.nn.embedding_lookup(self.item_embeddings, self.i)
self.i_bias = tf.gather(self.item_bias, self.i)

self.pre_logits = tf.reduce_sum(tf.multiply(self.u_embedding, self.i_embedding), 1) + self.i_bias
self.pre_loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=self.label,
                                                        logits=self.pre_logits) + self.lamda * (
    tf.nn.l2_loss(self.u_embedding) + tf.nn.l2_loss(self.i_embedding) + tf.nn.l2_loss(self.i_bias)
)

d_opt = tf.train.GradientDescentOptimizer(self.learning_rate)
self.d_updates = d_opt.minimize(self.pre_loss, var_list=self.d_params)
```

Generator

```
rating = sess.run(generator.all_logits, {generator.u: u})
exp_rating = np.exp(rating)
prob = exp_rating / np.sum(exp_rating)  # prob is generator distribution p_\theta

pn = (1 - sample_lambda) * prob
pn[pos] += sample_lambda * 1.0 / len(pos)
# Now, pn is the Pn in importance sampling, prob is generator distribution p_\theta

sample = np.random.choice(np.arange(ITEM_NUM), 2 * len(pos), p=pn)
########################################################################
# Get reward and adapt it with importance sampling
########################################################################
reward = sess.run(discriminator.reward, {discriminator.u: u, discriminator.i: sample})
reward = reward * prob[sample] / pn[sample]
########################################################################
# Update G
########################################################################
_ = sess.run(generator.gan_updates,
             {generator.u: u, generator.i: sample, generator.reward: reward})
```

Generator

```
self.u_embedding = tf.nn.embedding_lookup(self.user_embeddings, self.u)
self.i_embedding = tf.nn.embedding_lookup(self.item_embeddings, self.i)
self.i_bias = tf.gather(self.item_bias, self.i)

self.reward_logits = tf.reduce_sum(tf.multiply(self.u_embedding, self.i_embedding),
                                   1) + self.i_bias
self.reward = 2 * (tf.sigmoid(self.reward_logits) - 0.5)
```

Generator

```
self.all_logits = tf.tensordot(self.u_embedding, self.item_embeddings,1) #+ self.item_bias

self.gan_loss = -tf.reduce_mean(tf.log(self.all_logits) * self.reward) + self.lamda * (
    tf.nn.l2_loss(self.u_embedding) + tf.nn.l2_loss(self.i_embedding) + tf.nn.l2_loss(self.i_bias))

g_opt = tf.train.GradientDescentOptimizer(self.learning_rate)
self.gan_updates = g_opt.minimize(self.gan_loss, var_list=self.g_params)
```

References

[1] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A Neural Autoregressive Approach to Collaborative Filtering. In Proceedings of the 33rd International Conference on Machine Learning - Volume 48 (ICML'16). JMLR.org, 764–773.

[2] Larochelle, Hugo, and Iain Murray. "The neural autoregressive distribution estimator." *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011.

[3] Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering." *Proceedings of the 24th international conference on Machine learning*. ACM, 2007.

[4] Uria, Benigno, et al. "Neural autoregressive distribution estimation." *Journal of Machine Learning Research* 17.205 (2016): 1-37.

[5] Uria, Benigno, Iain Murray, and Hugo Larochelle. "A deep and tractable density estimator." *International Conference on Machine Learning*. 2014.

[6] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval (2017)