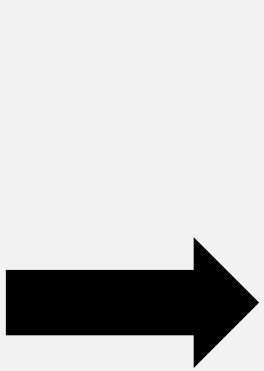
Sequential Recommendation System Using Transformer Architecture

– 트랜스포머를 이용한 시퀀셜 추천시스템 –

202150294 오 형 택

Datascience LAB

Sequential Recommendation System



<실제 사용자가 구매한 아이템>

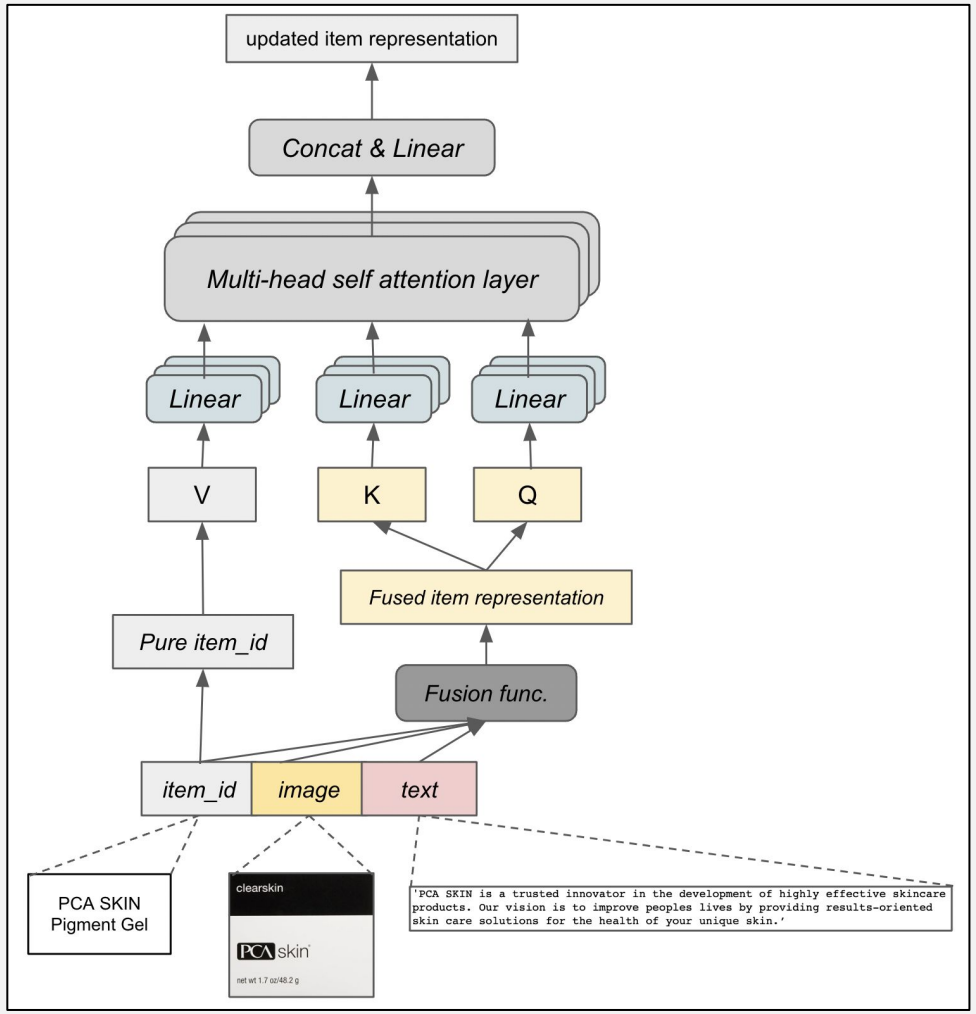
- ✔ 실제로 사용자가 아이템을 구매할 때 이미지와 텍스트 같은 부가적인 정보가 중요할 것이라고 가정함
- ✔ 사용자의 행동패턴을 학습 시, 아이템의 이미지(pre-trained VGG)와 텍스트(pre-trained BERT) 정보를 함께 반영

Data Design

	# users	# items	# average length	# side information
Luxury Beauty	3,362	1,494	7.1	Image, Description
Sports and Outdoors	153,940	55,697	5.7	Image, Description

- ✔ Amazon Dataset : Luxury&Beauty, Sports&Outdoors
- ✔ 사용자 혹은 아이템 구매이력 5건 이하 제거
- ✔ 이미지 및 텍스트 정보 없는 아이템 데이터셋 제거
- ✔ Test : 사용자의 마지막 구매 아이템 (n)
- ✔ Valid : 사용자의 마지막 구매 이전 아이템 (n-1)

From Transformer



user_seq

user_pos_seq

user_neg_seq

```
tensor([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,  133,  134,  170,  284,  284,  909,  984,
         1213, 1216],
        [   0,    0,    0,    0,    0,    0,    0,    8,   36,  186,  361,  402,
          410,  454,  361,  402,  410,  454,  521,  574,  578,  580,  624,  722,
          761,  830,  851,  859,  868,  870,  872,  950,  986,  999, 1004, 1032,
         1084, 1165, 1290, 1294, 1300, 1317, 1348, 1390, 1410, 1440, 1463,  124,
          246,  665],
        [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,  123,  132,
          133,  467]], dtype=torch.int32)
```

```
tensor([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,  132,  133,
          467,  467]], dtype=torch.int32)
```

```
tensor([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,  826,  318,
          425,  598]], dtype=torch.int32)
```

- ✓ user_seq, user_pos_seq, user_neg_seq ∈ $\mathbb{R}^{batch\_size \times \max\_len}$
- ✓ Image_feature ∈ $\mathbb{R}^{batch\_size \times \max\_len \times 4096}$
- ✓ text_feature ∈ $\mathbb{R}^{batch\_size \times \max\_len \times 768}$

참고 : Non-invasive Self-attention for Side Information Fusion in Sequential

From Transformer

```python
def compute_relevance_scores(self, item_emb , q_items):
    q_emb = self.ie(q_items)
    out = (item_emb*q_emb).sum(dim=-1)

    return out

def training_step(self, batch, batch_idx):
    u, seq, pos, neg, image_feature, text_feature = batch

    item_emb = self.forward(seq, image_feature, text_feature)
    pos_scores = self.compute_relevance_scores(item_emb, pos)
    neg_scores = self.compute_relevance_scores(item_emb, neg)

    pos_labels = torch.ones(pos_scores.shape, device=self.device)
    neg_labels = torch.zeros(neg_scores.shape, device=self.device)

    indices = torch.where(pos!=0)

    loss = self.loss(pos_scores[indices], pos_labels[indices]) +\
    self.loss(neg_scores[indices], neg_labels[indices]) \
    + self.hparams.l2_pe_reg*torch.linalg.matrix_norm(next(self.pe.parameters()).data)

    self.log('loss', loss.item(), prog_bar=True, logger=True)

    return {'loss': loss}
```

Transformer Encoder

Binary Cross Entropy

```python
def _shared_val_step(self, batch, batch_idx):
    final_seq, val_test_seq, image_feature, text_feature = batch

    with torch.no_grad():
        input_emb = self.forward(final_seq, image_feature, text_feature)
        final_feat = input_emb[:, -1, :]
        val_test_emb = self.ie(val_test_seq)
        logits = torch.bmm(val_test_emb, final_feat.unsqueeze(-1))

        predictions = -logits.squeeze()
        GROUND_TRUTH_IDX = 0

        TOP_N = self.hparams.top_k
        _, indices = torch.topk(predictions, TOP_N, dim=1, largest=False)
        _, rank = torch.where(indices == GROUND_TRUTH_IDX)
        HITS = torch.as_tensor(rank <= TOP_N , dtype=torch.int)
        NDCG = HITS/torch.log2(rank+2)
    return HITS.sum().item()/len(final_seq), NDCG.sum().item()/len(final_seq)
```

마지막 구매 아이템 [0] + 구매하지 않은 아이템들 [1:]

상위 점수를 갖는 K개 아이템

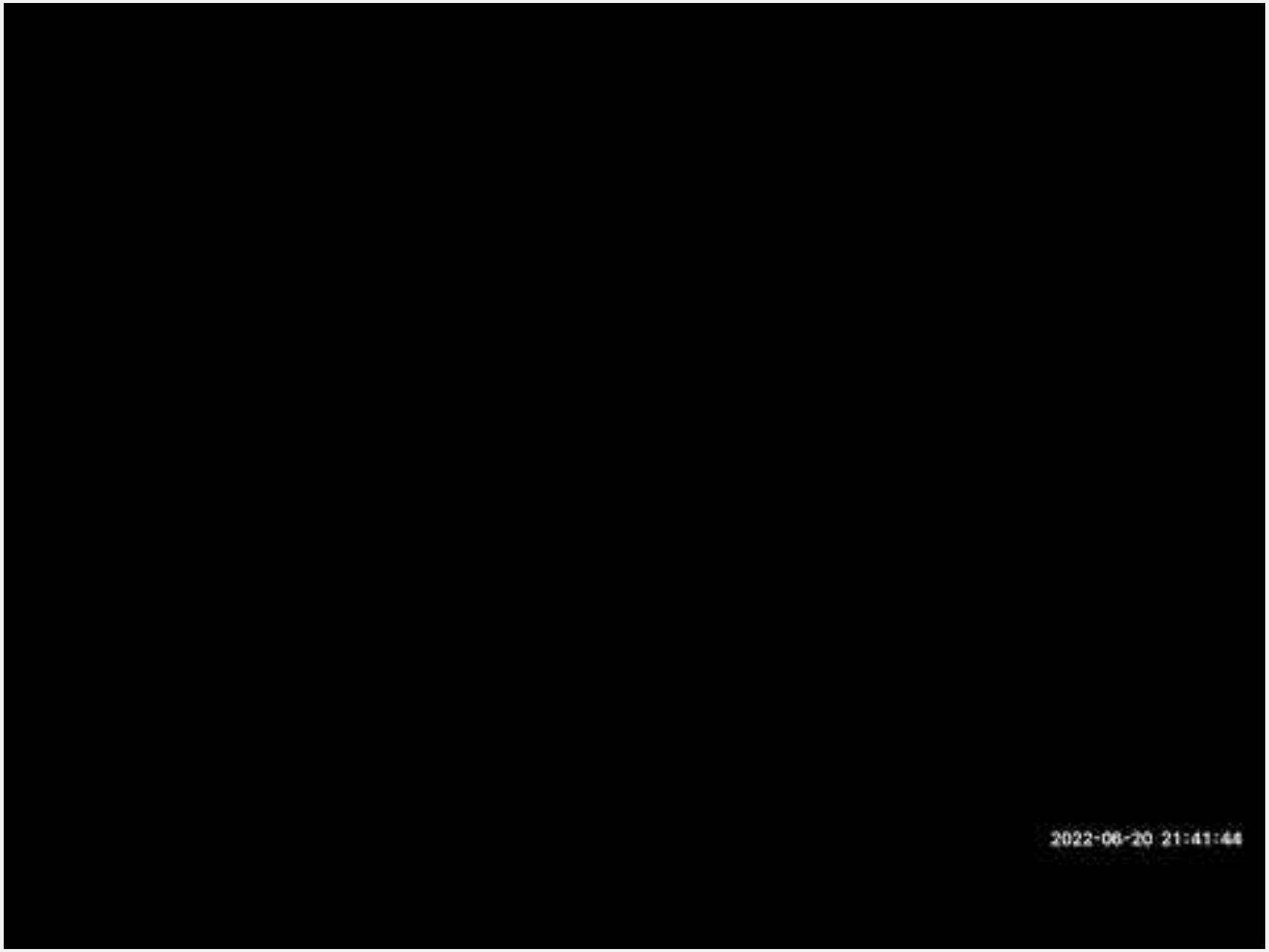
<사용자가 구매한 아이템>

<상위 추천 아이템>

Experiment Result

We hypothesize that with a more affluent corpus, the models are possible to learn good enough item embeddings even from the item context itself, leaving a smaller space for side information to make supplements.

	Luxury Beauty		Sports and Outdoors	
	Hit@10	NDCG@10	Hit@10	NDCG@10
SASRec	0.4763	0.3496	0.2569	<u>0.1428</u>
SASRecNOVA (sum)	<u>0.5261</u>	**0.363**	<u>0.2578</u>	0.1424
SASRecNOVA (concat)	0.4982	0.348	0.2498	0.1417
SASRecNOVA (gating)	**0.5264**	<u>0.3598</u>	**0.2585**	**0.1498**



https://youtu.be/aBm_zfzngJ0

Q & A