[Paper Review] Wide & Deep Learning for Recommender Systems

Data Science & Artificial Intelligence Lab

2024.01.22 구본우

ISysE
KAIST
Industrial & Systems Engineering

CONTENT

ABSTRACT

- Paper in 2016 by Google
- Google Play mobile app store recommendation
- To combine benefits of memorization and generalization for recommender systems
- Memorization by wide linear models
- Generalization by deep neural networks

INTRODUCTION

- One challenge is to achieve both memorization and generalization
- Memorization: learning the co-occurrence of items and exploiting the correlation in the historical data
  - By Cross product transformation of one hot vector
  - Eg. AND(User=구본우, Item=맥북) → 1, new binary feature
- Generalization: transitivity of correlation and explores new feature combinations that never occurred
  - By adding features that are less specific
  - Eg. AND(User installed category = video, impressed category = music) but with manual feature engineering

INTRODUCTION

- Wide Model (Memorization)
  - Simple, scalable, interpretable linear model
  - Logistic regression
  - One-hot encoding
  - Cross product
  - Perform well in sparse data

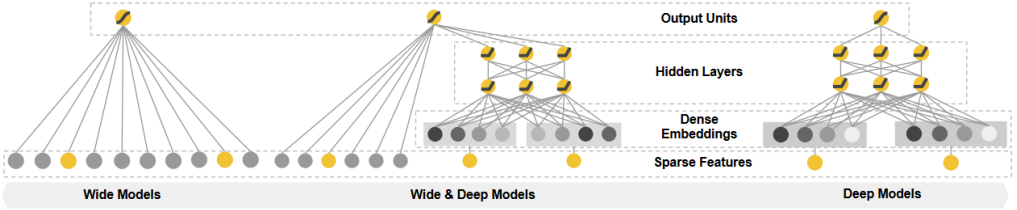


Figure 1: The spectrum of Wide & Deep models.

- Deep Model (Generalization)
  - Embedding based models: Deep neural network, Factorization Machines
  - By learning a low-dimensional dense embedding vector for each query and item feature
  - Difficulty in learning if data is sparse → can lead to over-generalize
- **Wide + Deep model (Memorization + Generalization)**
  - Jointly training feed-forward neural networks with embeddings and linear model with feature transformations for generic recommender systems with sparse inputs

RECOMMENDER SYSTEM OVERVIEW

1. A query is generated when a user visits the app store
2. The retrieval system returns a short list of items that best match the query by machine-learned model
3. Ranking system ranks all items by score
    1. P(*y* l *x*): the probability of a user action *y* give the features *x*
4. User performs certain actions such as clicks or purchases and these are recorded as logs and used for training

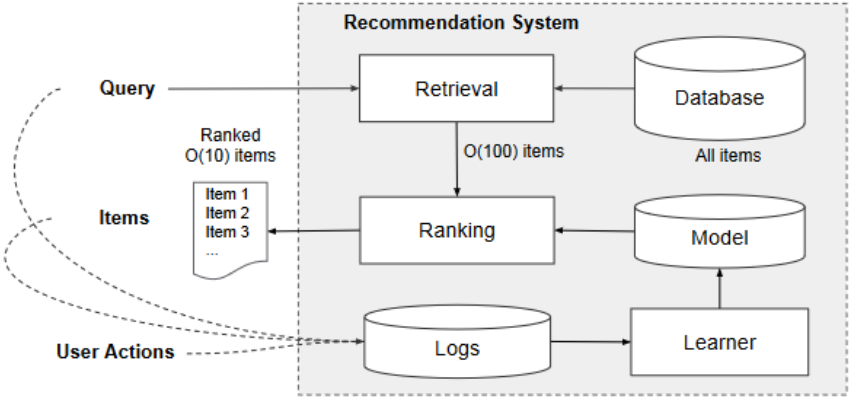
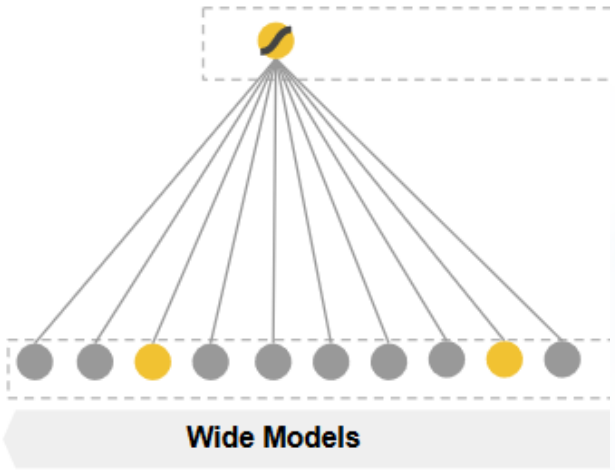


Figure 2: Overview of the recommender system.

WIDE COMPONENT

- Wide component: generalize linear model of $y = w^T x + b$
  - y: prediction
  - x: [x1, x2, …, xd] feature
  - w: [w1, w2, …, wd] weight
  - b: bias
- Feature set: raw input features + transformed features
  - Cross-product transformation

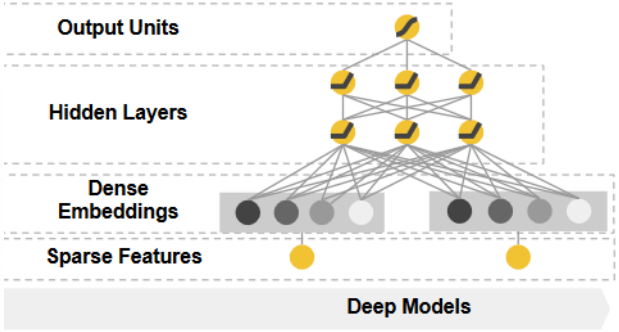
$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

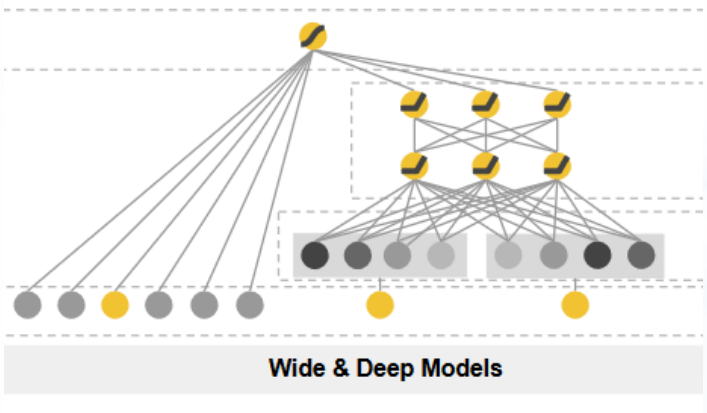


  - Eg. X = [User, Item, Age], $\phi_k(x)$: k-th cross product feature that considers User and Age
  - Then, $C_{kUser} = C_{kAge} = 1$ and $C_{kItem} = 0 \longrightarrow \phi_k(x) = x^1_{User} x^0_{Item} x^1_{Age}$
  - Captures interactions between binary features and adds non-linearity to the generalized linear model

DEEP COMPONENT

- Feed Forward Neural Network
- High dimensional categorical features converted into low dimensional and dense real-valued vector = embedding vector
- Concatenate with continuous features
- Embedding vectors are initialized randomly and values are trained to minimize the loss function
- Each hidden layer performs $a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)})$
  - a: activation
  - b: bias
  - W: model weights
  - f: activation function
  - l : layer number





JOINT TRAINING OF WIDE & DEEP MODEL

- Wide and deep component are combined using weighted sum
- Fed to one common logistic loss function: joint training
  - Ensemble (X): models are trained separately
- Model size
  - Ensemble: each individual model size needs to be larger
  - Joint training: wide part only needs to complement the weakness of deep part with a small number of feature transformations

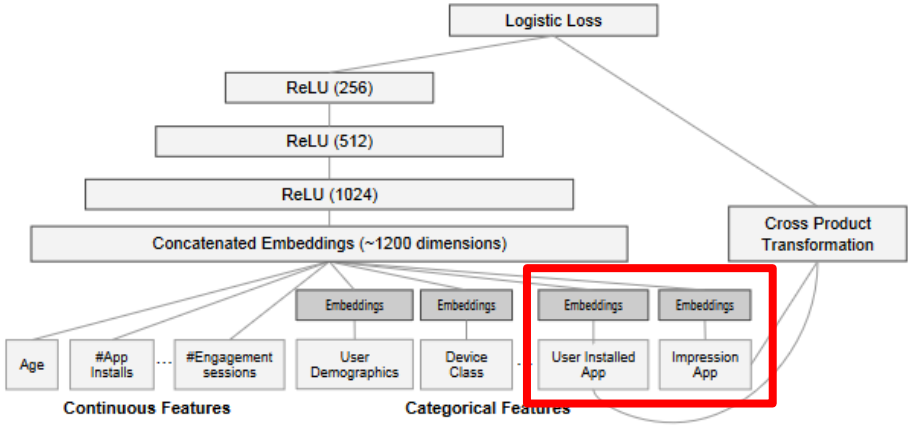


Figure 4: Wide & Deep model structure for apps recommendation.

JOINT TRAINING OF WIDE & DEEP MODEL

- Back propagation using mini batch stochastic optimization
- Follow-the-regularized-leader (FTRL) algorithm with L1 regularization as the optimizer for wide part
- AdaGrad for the deep part
- Model's prediction

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

  - Y: binary class label
  - σ: sigmoid function
  - $\phi_k(x)$: cross product transformation
  - w_wide: vector of all wide model weights
  - w_deep: weights applied on the final activation a^{l_f}

SYSTEM IMPLEMENTATION

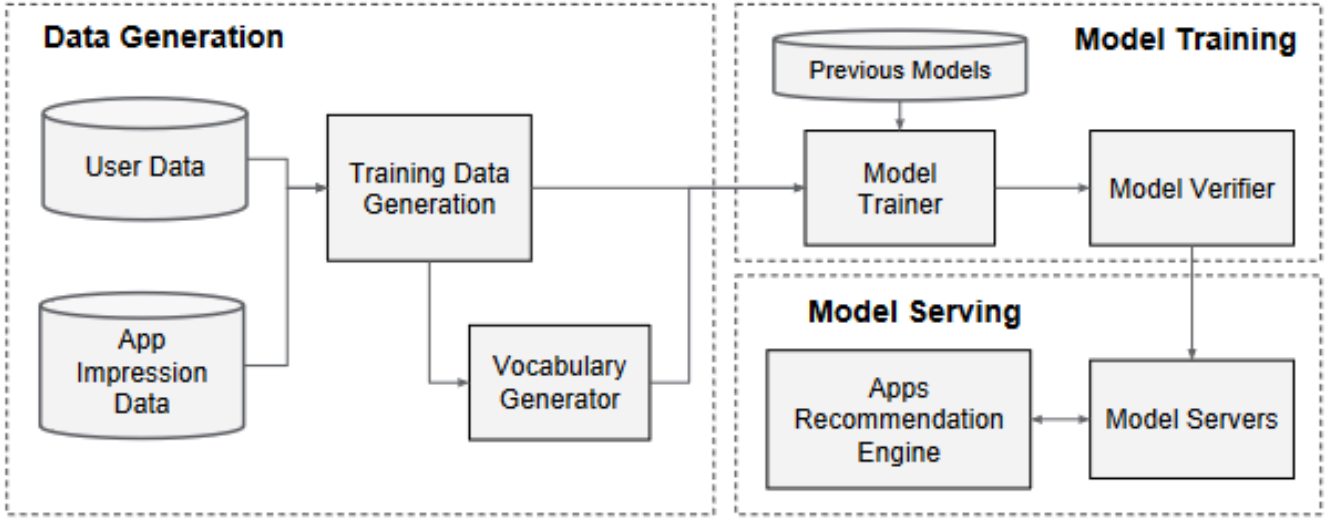
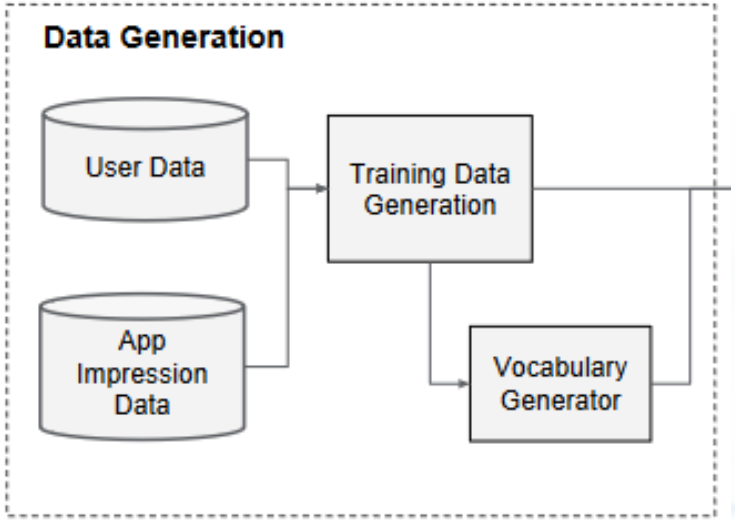


Figure 3: Apps recommendation pipeline overview.

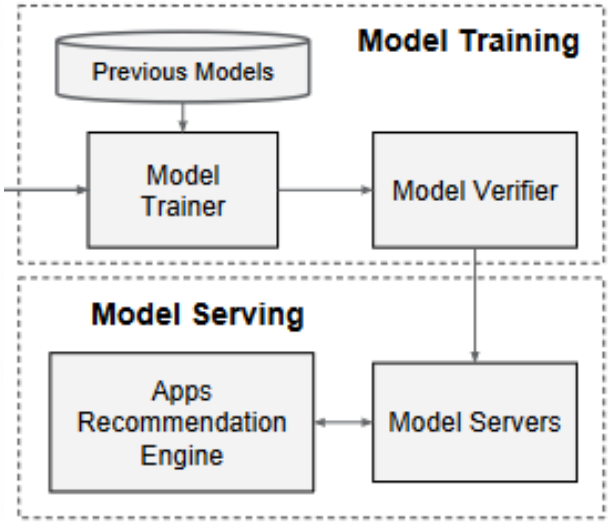
DATA GENERATION

- User and app impression data are used to generate training data
- App acquisition:
  - 1 if the impressed app was installed
  - 0 otherwise
- Categorical features are mapped to integer IDs
- Continuous real-valued features are normalized to [0,1]



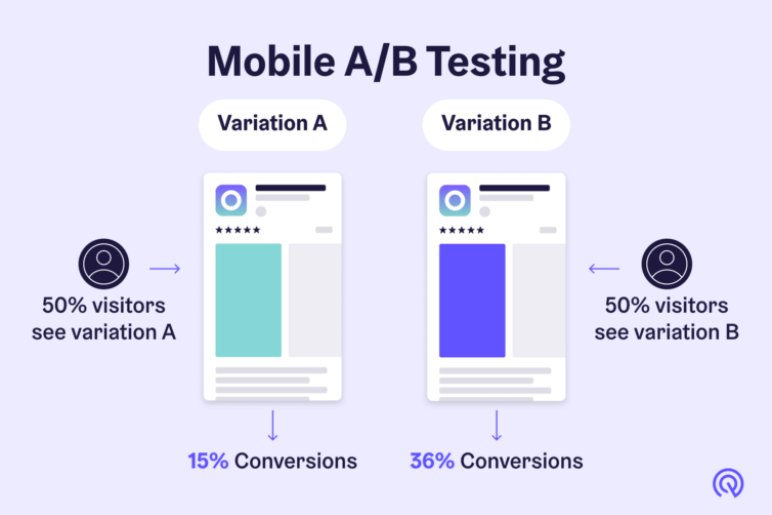
MODEL TRAINING AND SERVING

- Input layer takes in training data and vocabularies and generate sparse and dense feature
- Embedding vector is learned for each categorical feature
- Concatenate all the embeddings with dense features
- Concatenated vector is then fed into 3 ReLU layers and finally through logistic function
- Warm-starting system: initialize new model with previous weights and embeddings
- Optimize the performance using multithreading parallelism by running smaller batches



EXPERIMENT RESULTS

- A/B Testing for 3 weeks
- A randomized experimentation process wherein two or more versions of a variable are shown to different segments of website visitors at the same time to determine which version leaves the maximum impact and drives business metrics
- Which model gives the maximum online acquisition gain!



EXPERIMENT RESULTS

Table 1: Offline & online metrics of different models. Online Acquisition Gain is relative to the control.

Model	Offline AUC	Online Acquisition Gain
Wide (control)	0.726	0%
Deep	0.722	+2.9%
Wide & Deep	0.728	+3.9%

- Highest online acquisition gain for Wide & Deep model
- More significant impact on online
  - Impressions and labels are fixed in offline
  - Online system can generate new exploratory recommendations by blending memorization and generalization

SERVING PERFORMANCE

Table 2: Serving latency vs. batch size and threads.

Batch size	Number of Threads	Serving Latency (ms)
200	1	31
100	2	17
50	4	14

- Better performance in multithreading!

CONCLUSION

- Idea of combining linear models with cross product transformations and deep neural networks with dense embeddings
  - Inspired by factorization machines which add generalizations to linear model by factorizing the interactions
  - Wide & deep expands the model capacity by learning highly non linear interactions through neural networks instead of dot product
- Effectively memorize and generalize!
- Greater online acquisition gain in Google Play App Store

IMPLEMENTATION-FEATURE PROCESSING

- Data – MovieLens 1m
  - Users
  - Movies
  - Ratings
- Categorical Data One Hot Encoding
- Cross product transformation
  - Age
  - gender

```
data_wide
```

✓ 0.1s Python

Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	...	Western	female	male	adult	youth	female-adult	female-youth	male-adult	male-youth	ratings
0	1	1	1	0	0	0	...	0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1
0	1	1	1	0	0	0	...	0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1
0	1	1	1	0	0	0	...	0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1
0	1	1	1	0	0	0	...	0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1
0	1	1	1	0	0	0	...	0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	0	0	0	0	1	...	0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1
0	0	0	0	0	0	1	...	0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0
0	0	0	0	0	0	1	...	0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1
0	0	0	0	0	0	1	...	0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0
0	0	0	0	0	0	1	...	0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1

IMPLEMENTATION - MODEL

```
class WideAndDeep(nn.Module):

   def __init__(self, data_wide, data_deep):
      super(WideAndDeep, self).__init__()

      # Deep Model
      self.embed_gender = nn.Embedding(num_embeddings=len(np.unique(data_deep.gender)), embedding_dim=8)
      self.embed_age = nn.Embedding(num_embeddings=len(np.unique(data_deep.age)), embedding_dim=8)
      self.embed_occupation = nn.Embedding(num_embeddings=len(np.unique(data_deep.occupation)), embedding_dim=8)
      # As gender, age, occupation column will be replaced with embeddings
      self.len_genres = len(data_deep.columns) - 3

      self.Linear_1 = nn.Linear(in_features = 24 + self.len_genres, out_features = 64)
      self.Linear_2 = nn.Linear(in_features = 64, out_features = 32)
      self.Linear_3 = nn.Linear(in_features = 32, out_features = 16)

      # WideAndDeep Model
      self.Linear = nn.Linear(in_features = len(data_wide.columns)+ 16, out_features = 1)
      self.Logistic = nn.Sigmoid()
```

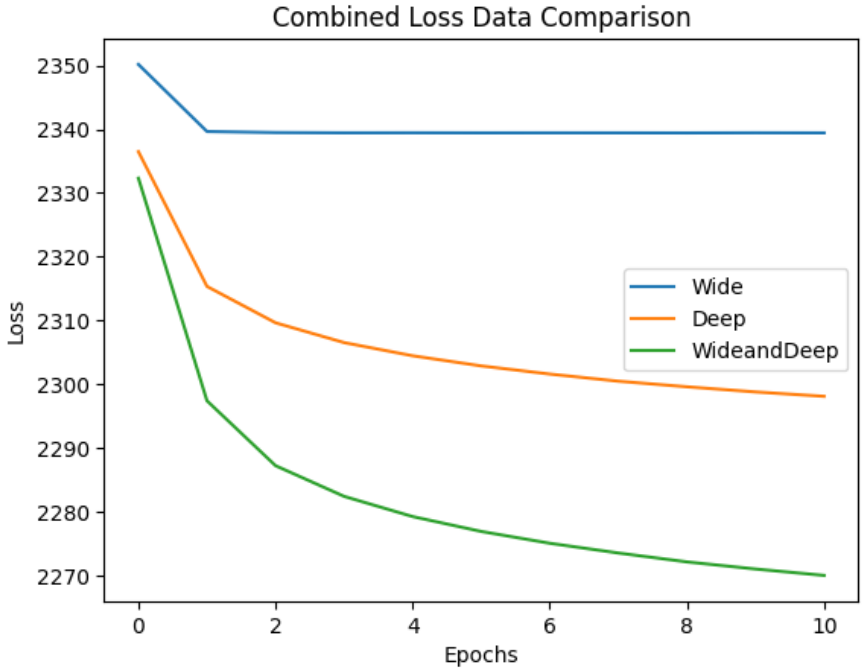
IMPLEMENTATION - MODEL

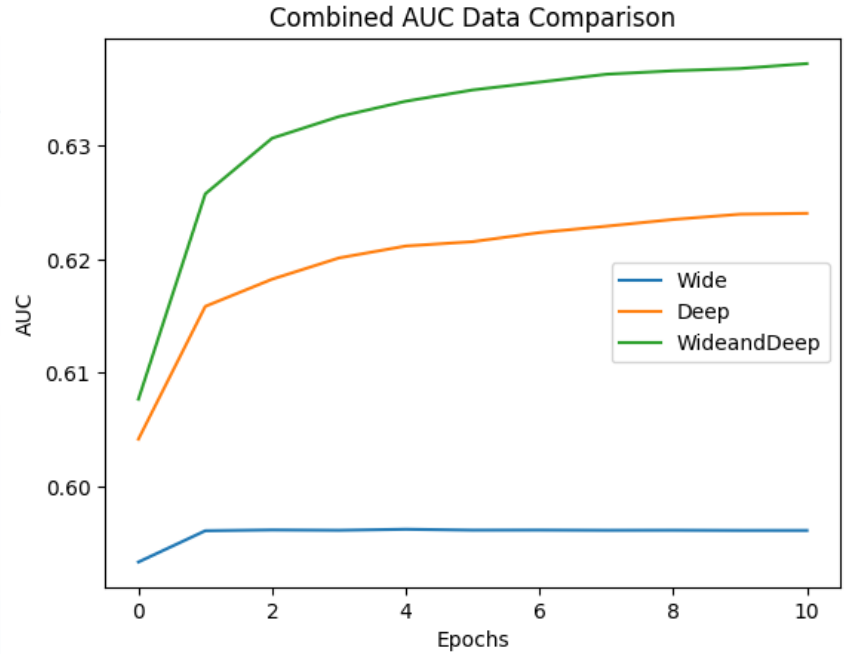
```
def forward(self, X_wide, X_deep):

    # Deep Model
    # Embedding genres except gender, age, occupation feature
    embedding_genres = X_deep[:,:-3]
    # third from the last is gender column
    embedding_gender = self.embed_gender(X_deep[:,-3])
    # second from the last is age column
    embedding_age = self.embed_age(X_deep[:,-2])
    # last column is occupation
    embedding_occupation = self.embed_occupation(X_deep[:,-1])

    input_deep = torch.cat([embedding_genres, embedding_gender, embedding_age, embedding_occupation], dim = 1)

    input_deep = self.Linear_1(input_deep)
    input_deep = nn.ReLU()(input_deep)
    input_deep = self.Linear_2(input_deep)
    input_deep = nn.ReLU()(input_deep)
    input_deep = self.Linear_3(input_deep)
    output_deep = nn.ReLU()(input_deep)

    # WideAndDeep
    input_wad = torch.cat([X_wide,output_deep], dim = 1)
    logits = self.Linear(input_wad)
    out = self.Logistic(logits)

    return out
```

EXPERIMENT RESULT

```
Iteration : 1 -- Total loss : 2332.3230, Test accuracy : 0.608, Time : 6.8174
Iteration : 10 -- Total loss : 2297.4150, Test accuracy : 0.626, Time : 66.2853
Iteration : 20 -- Total loss : 2287.2385, Test accuracy : 0.631, Time : 244.3209
Iteration : 30 -- Total loss : 2282.4253, Test accuracy : 0.633, Time : 339.9496
Iteration : 40 -- Total loss : 2279.2559, Test accuracy : 0.634, Time : 404.7947
Iteration : 50 -- Total loss : 2276.9238, Test accuracy : 0.635, Time : 470.0879
Iteration : 60 -- Total loss : 2275.0703, Test accuracy : 0.636, Time : 535.4395
Iteration : 70 -- Total loss : 2273.5425, Test accuracy : 0.636, Time : 600.5773
Iteration : 80 -- Total loss : 2272.1487, Test accuracy : 0.637, Time : 666.0782
Iteration : 90 -- Total loss : 2271.0332, Test accuracy : 0.637, Time : 731.4983
Iteration : 100 -- Total loss : 2270.0298, Test accuracy : 0.637, Time : 795.0593
```

EXPERIMENT RESULT





느낀점

- Better performance than wide or deep model
- Good understand the general flow of recommender system pipeline
- Computationally expensive
- Not popularly used anymore(?) → Neural Collaborative Filtering, DeepFM