Stats AI

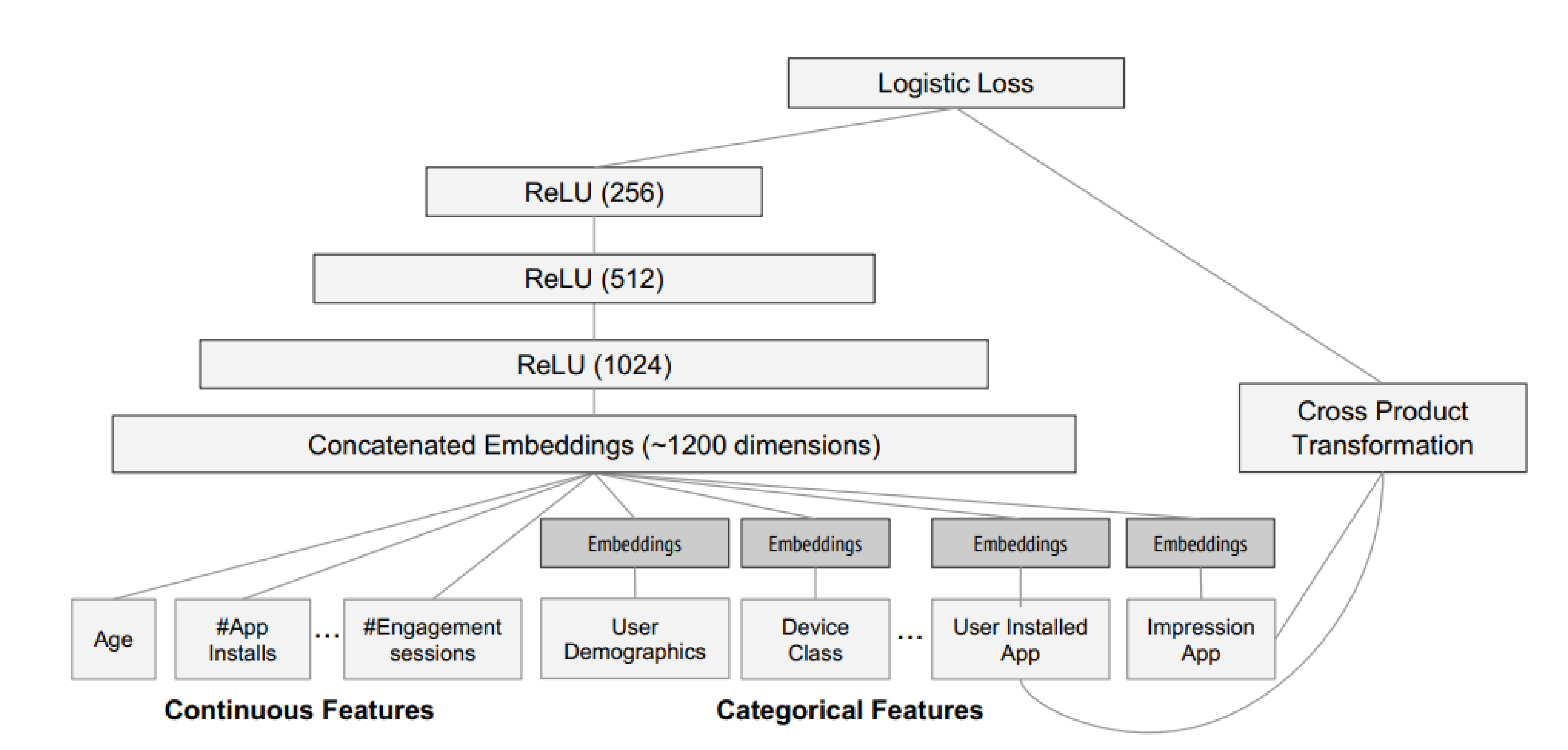
Deep Learning Recommender Systems

Author: Hussain Abbas, MSc

Wide & Deep Learning 2016

Stats AI

Wide & Deep Learning for Google Play Store Recs



https://dl.acm.org/doi/pdf/10.1145/2988450.2988454

Stats AI

Example

- Query:
  - Suppose we have a food delivery app that allows a user to state what kind of food that they are craving
  - Their statement is the "query"
  - Example: "I want seafood"

- Item:
  - The app predicts the dish the user will like best and delivers it to the user
  - The dish is the "item"
  - Example: "Shrimp fried rice"

- Online Evaluation Criteria:
  - If a dish was consumed by the user, we record a score of 1, otherwise, it's 0
  - This 0 or 1 is called the "label"

Stats AI

Visualizing the Data

query	item predicted	label
seafood	shrimp fried rice	0
seafood	fried shrimp	1
chicken	buffalo wings	1

- Here we show three separate transactions

- Each transaction consists of a query, item, and label
  - Query: what the user said they wanted
  - Item: the recommendation based upon the user's query
  - Label: whether the user ended up liking the item that was recommended

- The larger the data set the better our understanding of users tastes and preferences

Stats AI

MVP 1.0

- Suppose we want to develop an MVP
- An MVP consists of several releases. In every release the product gets better
- First release: character match recommender system:
  - When a user says they are craving “X”, the system will select the item with the most similar text string to X
  - Thus, if a user says they are craving “fried chicken”:
    - The system may show them “chicken fried rice”
    - Why? Because “chicken” and “fried” are in both strings
- Bottomline:
  - Obviously, chicken fried rice and fried chicken are very different dishes
  - Thus, users get “irrelevant” recommendations

query	item predicted	label
seafood	seafood platter	1
fried chicken	chicken fried rice	0

Stats AI

MVP 2.0

- Second release: Wide recommender system:
  - MVP produces irrelevant recs -> Apply ML to try to increase relevancy
  - We can learn an ML classifier from the dataset obtained from MVP 1.0
  - Input features: query, item (what we're feeding into the ML model)
  - Output label: whether they consumed the product or not (1 or 0)
  - The ML model is "trained" to predict the probability of consumption given the query, item pair

- The ML model is trained, and we investigate the features:
  - The feature AND(query="fried chicken", item="chicken and waffles") is a huge win
    - meaning whenever users get this rec, they end up consuming it
  - The feature AND(query="fried chicken", item="chicken fried rice") is a dud
    - meaning even though the character match is higher, users don't consume it

- Bottomline:
  - The ML model learns to memorize what users like and dislike
  - It learns to recommend items based upon probabilities, i.e., when users say they want fried chicken, they don't want chicken fried rice, and won't eat it if they get it
  - This increases the relevancy of our recommendations for our users

Stats AI

MVP 3.0

- Third release: Deep Learning recommender system:
  - MVP 2.0 produces relevant but boring recs-> Apply DL to produce more interesting recs
  - We can learn a DL classifier from the dataset obtained from prior MVP versions

- The DL model learns takes the query, item pairs and converts them into low dimensional dense representations (called embedding vectors)

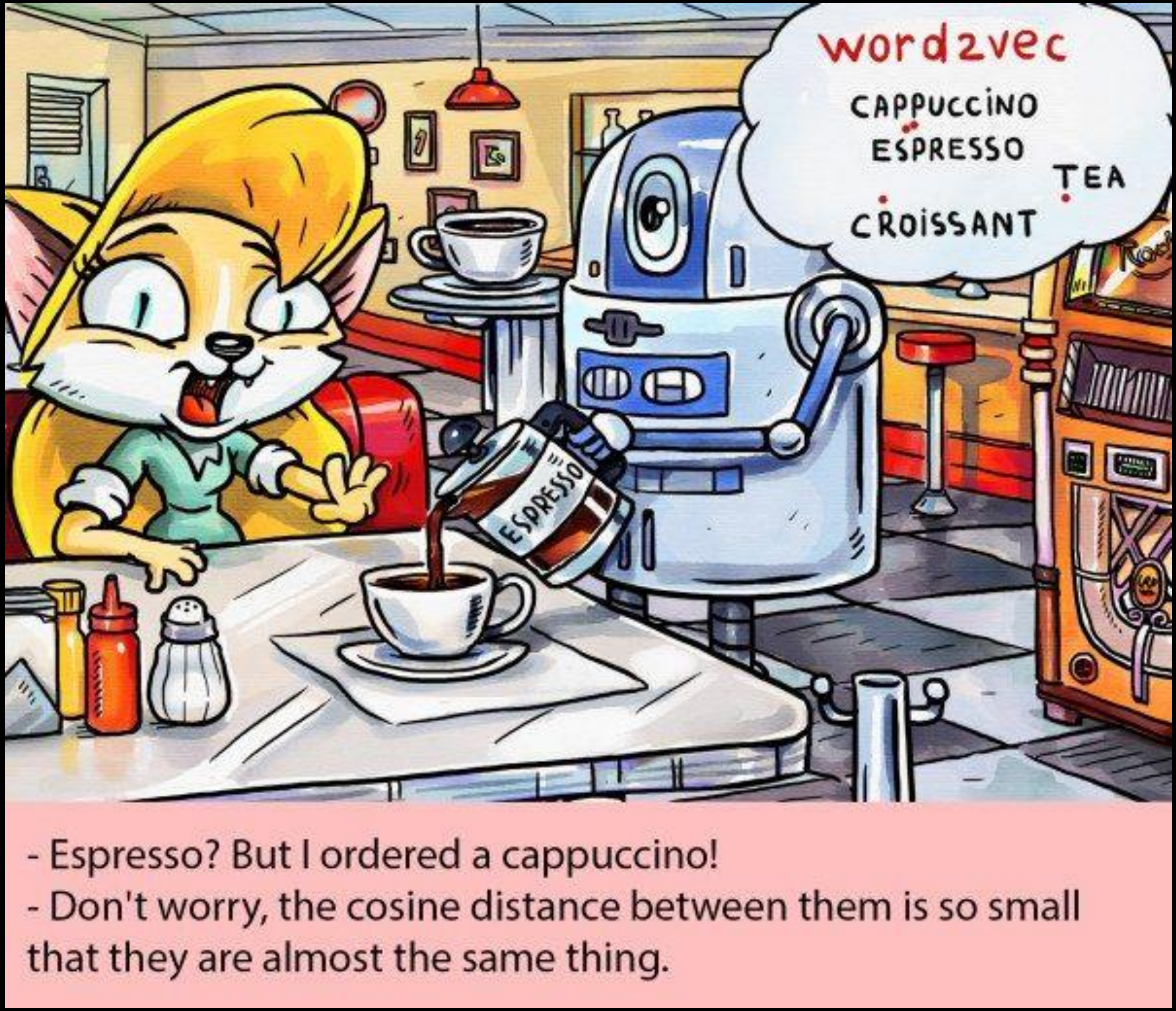
- These embedding vectors enables the DL model to generalize (deliver more interesting recs) by matching items to queries that are close in vectors space:

- Now suppose “fried chicken” and “burgers” are close to each other in the embedding space
  - This means that “fried chicken” and “burgers” are similar
  - Thus, users presenting a query of “fried chicken” are probably ok with getting “burgers”

- Bottomline:
  - The DL model improves upon MVP 2.0 by providing more interesting recs
  - Less work: the DL model performs automatic feature engineering on the raw data
  - The DL model learns the embedding vectors as part of the training process

Stats AI

MVP 4.0



Stats AI

MVP 4.0

- Fourth release: Wide & Deep Learning recommender system:
  - Problem: MVP 3.0 sometimes overgeneralizes (recommends irrelevant items)
  - We can learn a W&DL classifier from the dataset obtained from prior MVP versions

- Sometimes, a user has a very specific query for which there is no substitute:
  - A user states that they want an “iced decaf latte with nonfat milk”
  - In this case, it is true that its vector is close to “hot latte with whole milk”
  - Hence the issue with making recommendations based upon distance metrics
  - The solution is to model these niche specific queries with a wide linear model
  - Here, “wide” refers to the sparse matrices that result from one-hot encoding categorical variables with high cardinality

- Users making queries that are less specific like “seafood” or “italian food” may be more open to more generalization and discovering a diverse set of related items - > DL models excel here

- The DL model and the Wide model are jointly trained which results in them “specializing”

- Bottomline:
  - The W&DL model improves upon MVP 3.0 by better handling niche specific queries
  - The W&DL model combines the best of both worlds: DL with linear models

Stats AI

Technical Deep Dive

Stats AI

Background

- Given an input query (a set of user and contextual information), we want to:
  - find a set of relevant items in a database
  - rank the items based on certain objectives, such as clicks or purchases

- The goal is to strike a good balance between memorization and generalization:
  - Memorization: recommend things related to the things you've already clicked on
  - Generalization: recommend things you’ve never seen, i.e., novelty

- Bottomline:
  - Too much memorization -> the recommendations you get are boring
  - Too much generalization -> the recommendations you get are irrelevant
  - A good RS produces recs that are interesting (relevant and not boring)

Stats AI

Motivation

- Classification models such as Logistic Regression are often used to generate recs since they are easy to interpret

- In this case, we are using probability of click as a measure of relevance:
  - Probabilities lower than some threshold, say 90%, are deemed “irrelevant”
  - Ranking is simply sorting the probabilities in descending order
  - Ranking can also be done on “clicks” and “purchases”

- The input features into the classifier are often binary sparse features with one hot encoding:
  - "user_installed_app = netflix" has value 1 if user installed Netflix, 0 o/w

Stats AI

Motivation

- Memorization can be achieved via sparse cross-product features such as :
  - AND(user_installed_app = netflix, impression_app = "pandora")
  - has value 1 if user installed Netflix and then is later shown Pandora, 0 o/w

- Generalization can be achieved by using features that are <u>less granular</u>:
  - AND(user_installed_category = video, impression_app_category = "music")
  - has value 1 if user installed a video app and then is later shown a music app
  - This approach requires manual feature engineering

Stats AI

Motivation

- Sparse cross-product features have one limitation:
  - cannot generalize to query-item feature pairs that have not appeared in the training data

- Embedding-based models such as deep neural networks:
  - can generalize to previously unseen query-item feature pairs
  - They do so by learning a low-dimensional dense embedding vector for each query-item feature pair
  - This approach performs automatic feature engineering

- However, it is difficult to learn low-dimensional representations when the underlying query-item matrix is sparse and high rank:
  - This occurs when users have very specific preferences
  - This occurs when we have niche items with very narrow appeal
  - This results in irrelevant recommendations due to over-generalization

Stats AI

Motivation

- Linear models with cross-product feature transformations can memorize these exception rules with fewer parameters.

- Thus, we can achieve both memorization and generalization in one model by jointly training a linear model component and a neural network component

- Bottomline:
  - DNN’s improve upon Logistic Regression in their ability to perform automatic feature engineering on sparse binary features as well as their ability to generalize to previously unseen query-item feature pairs
  - However, DNN’s over generalize on sparse high rank matrices which occur when users have very niche preferences or items have small appeal
  - The solution is to model these edge cases with a linear model and then combine the two models to produce a final prediction

Stats AI

Results

- Wide linear models can effectively memorize sparse feature interactions using cross-product feature transformations

- Deep neural networks can generalize to previously unseen feature interactions through low dimensional embeddings

- The Wide & Deep learning rec sys framework combines the strengths of both types of model

- Google applied this method to Google Play, a large-scale app store:
  - Experiment results showed the Wide & Deep model led to a statistically significant increase in app installs over wide-only and deep-only models

- https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
- https://arxiv.org/abs/1606.07792

Stats AI