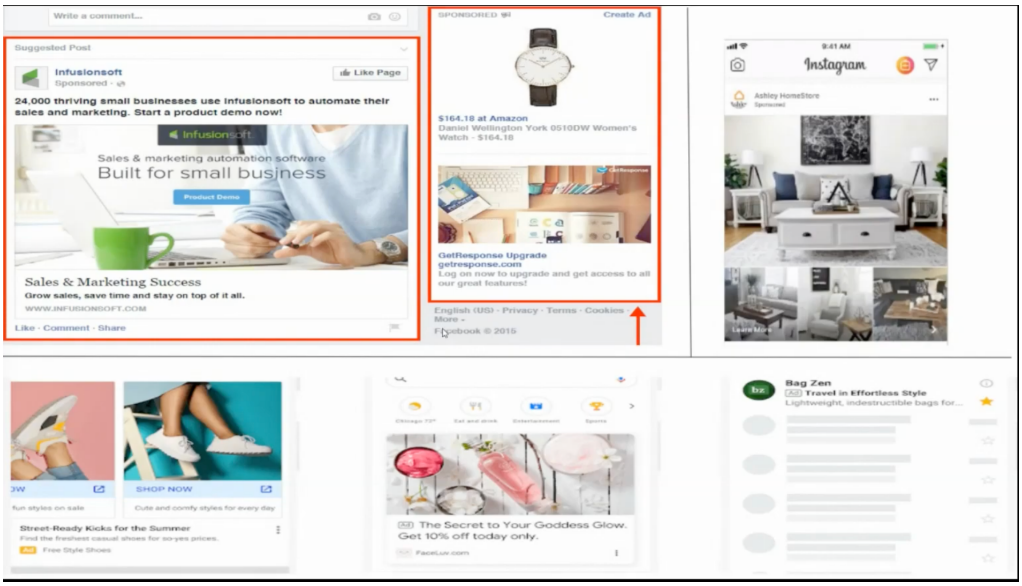
-ML System Design - Ad Click Predictions on Social Platforms

Video: https://www.youtube.com/watch?v=wxUx3gIUEsk

Business Objective:

Ad Clicks Prediction on Social Media Platform

Ads on Different Platforms



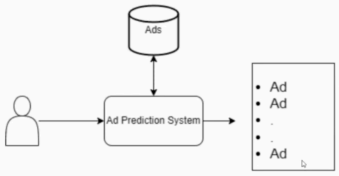
Clarifying requirements:

- To design an Ad Prediction System;
  - Business Objective - To maximize revenue.
- Assumptions and Constraints:
  - Ads on user feed.
  - No “Ad fatigue period” and “Block Advertiser” but have “hide this ad”.

- Is there an existing system already in place?
- What is the main objective: accuracy, latency?
- Is there labeled data? If so, how much
- Can we trust the quality of data coming in?
- Where the model will be deployed? [cloud, edge, etc]
- Are there any constraints or limitations on model complexity or size?
- What are the key success metrics for this ML system in the long term?

Framing the problem as an ML task: To correctly predict if the user will click the ad.

- Specifying input/output:



Data Preparation : Data Engineering: 1. Ad, 2. User, 3. User-Ad Interaction

Ad data:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
5	59.99	23	59761.56	226.74	Sharable client-driven software	Jamieberg	1	Norway	2016-05-19 14:30:17	0
6	88.91	33	53852.85	208.36	Enhanced dedicated support	Brandonstad	0	Myanmar	2016-01-28 20:59:32	0
7	66.00	48	24593.33	131.76	Reactive local challenge	Port Jefferybury	1	Australia	2016-03-07 01:40:15	1
8	74.53	30	68862.00	221.51	Configurable coherent function	West Colin	1	Grenada	2016-04-18 09:33:42	0
9	69.88	20	55642.32	183.82	Mandatory homogeneous architecture	Ramirezton	1	Ghana	2016-07-11 01:42:51	0

User:

ID	Username	Age	Gender	City	Country	Language	Timezone

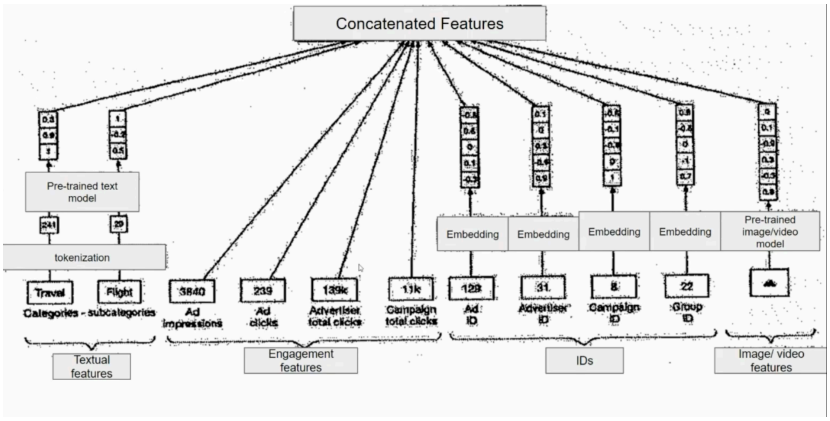
User Ad interaction data:

User ID	Ad ID	Interaction type	Dwell time[1]	Location (lat, long)	Timestamp
11	6	Impression	5 sec	38.8951 -77.0364	165845053
11	7	Impression	0.4 sec	41.9241 -89.0389	1658451365
4	20	Click	-	22.7531 47.9642	1658435948
11	6	Conversion	-	22.7531 47.9642	1658451849

Dwell time - total time an ad is present on user's screen.

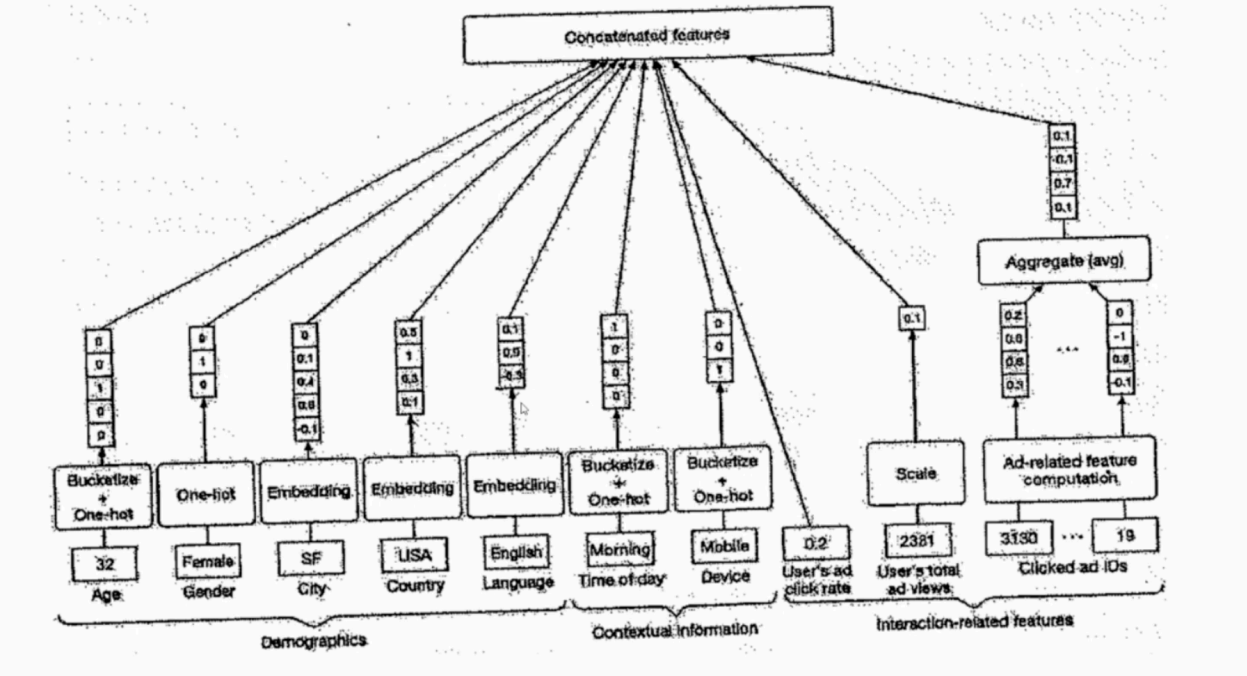
Data Preparation:
Feature Engineering:

- ID
- Image/Video
- Category and Subcategory
- Impression and click numbers
  - Total impressions/clicks on ad
  - Total impressions /clicks on ads supplied by an advertiser.
  - Total impressions of the campaign.



Data Preparation: Feature Engineering
User Features:

- Demographics : age, gender, city, country etc.
- Contextual Information : device, time of the day etc.
- Interaction-related features :
  - Clicked ads.
  - User’s historical engagement.



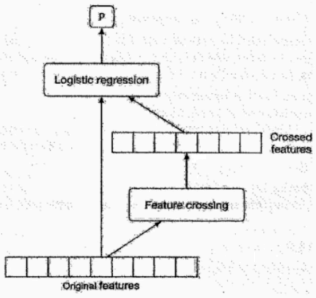
Model Development: Model Selection

- Logistic Regression (LR).
  - Non-linear problems can't be solved with linear decision boundaries.
  - Inability to capture feature interactions.

- Feature Crossing + LR.
  - f1: Country : [USA, China,England]
  - f2: Languages : [English, Chinese]

Country X Languages →

f3 : USA and English
f4 : USA and Chinese
f5 : England and English
f6 : England and Chinese
f7 : China and English
f8 : China and Chinese



FC + LR Shortcomings:

- Manual Process.
- Requires domain knowledge.
- Cannot process complex interactions.
- Sparsity.

User	Country (USA)	Country (China)	Country (England)	Language (English)	Language (Chinese)
1	1	0	0	1	0
2	0	1	0	0	1
3	0	0	1	1	0
4	1	0	0	0	1
5	0	1	0	1	0

User	USA & English	USA & Chinese	England & English	England & Chinese	China & English	China & Chinese
1	1	0	0	0	0	0
2	0	0	0	0	0	1
3	0	0	1	0	0	0
4	0	1	0	0	0	0
5	0	0	0	0	1	0

While automatically crossing *all* features removes the manual effort and domain knowledge requirement, it introduces these drawbacks:

- Combinatorial Explosion (Sparsity): The number of crossed features grows exponentially with the number of original features. This leads to extremely high-dimensional data, where most features are rarely seen (sparse data), making learning difficult and computationally expensive.
- Overfitting: With so many features, the model can easily overfit the training data, performing poorly on unseen data.
- Irrelevant Interactions: Not all feature combinations are meaningful. Crossing irrelevant features adds noise and can hinder performance.

Model: Step 1 – Candidate Generation

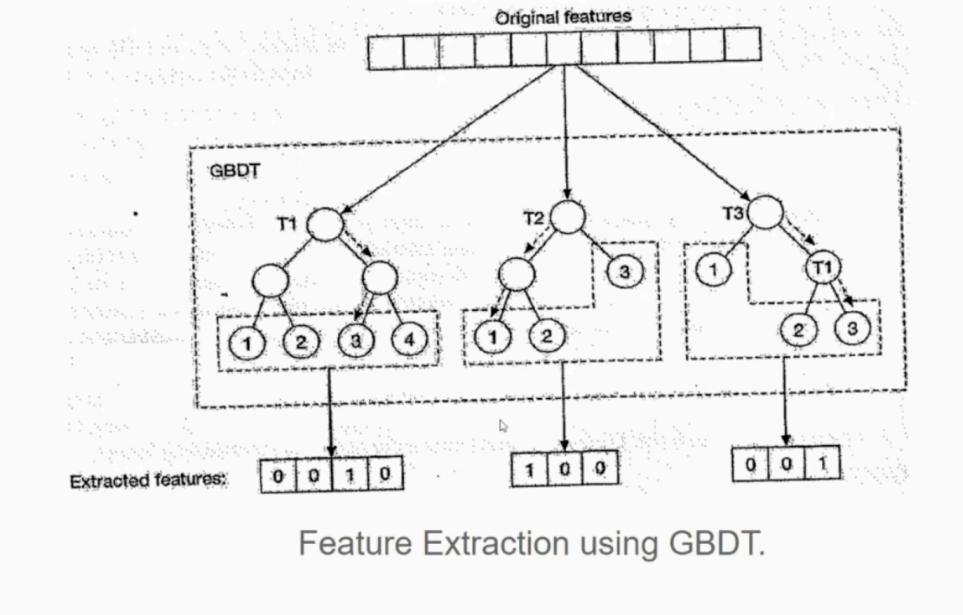
Here are some possible strategies for candidate generation in this context:

- **Rule-based filtering:** Use simple rules based on user demographics, interests, or context to filter the ad pool. For example:
    - Show ads for "running shoes" to users who have previously searched for "running gear."
    - Show ads for "local restaurants" to users based on their current location.
- **Keyword matching:** Match keywords in the user's query or profile with keywords associated with the ads. This is a common technique used in search advertising.
- **Content-based filtering:** If the ads have associated content (e.g., descriptions, images), use techniques like text similarity or image similarity to find ads that are relevant to the user's interests or the content they are viewing.

Model Development : Model Selection

- Gradient Boosted Decision Trees.
  - Pros : interpretable and easy to understand.
  - Cons :
    - Inefficient for continual learning.
    - Cannot train embedding layers.
- Gradient Boosted Decision Tree + Logistic Regression.
  - 1. To train the GBDT model to learn the task.
  - 2. Use feature selection and feature extraction and serve them as input to to LR model for predicting clicks.

GBDTs train on the entire dataset at once, sequentially adding trees. Continual learning introduces new data incrementally. Retraining GBDTs on combined old and new data causes "catastrophic forgetting" (loss of prior knowledge). Retraining is computationally expensive. Methods like incremental tree updates exist but are complex and not a complete solution. Other models (e.g., neural networks with continual learning strategies) are generally better for continual learning.



Feature Extraction using GBDT.

This diagram illustrates how Gradient Boosted Decision Trees (GBDTs) can be used for *feature extraction* or *feature transformation*. It's not showing a typical GBDT used for prediction but rather using the structure of the trees to create new features.

Here's a breakdown:

1. **Original Features:** The top row represents the original input features. These are the raw data that you would normally feed into a machine learning model.
2. **GBDT Ensemble:** The large box labeled "GBDT" represents an ensemble of decision trees (T1, T2, T3 in this case).

3. **Path Traversal:** For each data point (represented by the original features), you traverse each tree in the ensemble. The data point starts at the root node of each tree and follows the branches based on the feature values until it reaches a leaf node.
4. **One-Hot Encoding of Leaf Nodes:** The key idea is that each leaf node in each tree is treated as a new feature. If a data point ends up in a particular leaf node, that corresponding feature is set to 1; otherwise, it's 0. This is effectively a one-hot encoding of the leaf nodes.
5. **Extracted Features:** The bottom rows show the extracted features. Each row represents the output for a single tree. The "0010," "100," and "001" are examples of the one-hot encoded leaf node outputs.

Example:

Let's say a data point has feature values that cause it to:

- End up in leaf node 3 of tree T1.
- End up in leaf node 1 of tree T2.
- End up in leaf node 2 of tree T3.

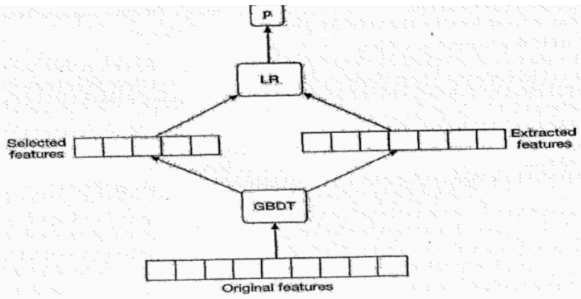
Then, the extracted features would be:

- T1: 0010 (1 at the 3rd position)
- T2: 100 (1 at the 1st position)
- T3: 001 (1 at the 2nd position)

These extracted features can then be concatenated to form a new feature vector that represents the original data point.

Why is this useful?

- **Non-linear feature transformations:** Decision trees can capture non-linear relationships between features. By using the leaf nodes as features, you are effectively transforming the original features into a new feature space that captures these non-linearities.
- **Improved performance with linear models:** These extracted features can then be used as input to linear models like logistic regression or linear SVMs, which might perform better on these transformed features than on the original features.
- **Dimensionality reduction (sometimes):** If the number of leaf nodes is smaller than the original feature space, this can also act as a form of dimensionality reduction.



Overview to GBDT + LR.

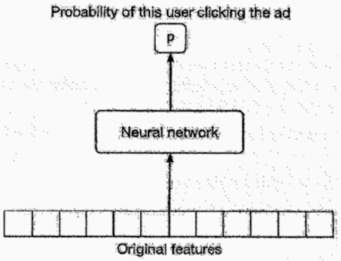
Pros:

- The features produced by GBDT have more predictive power making it easier for LR model to learn the task.

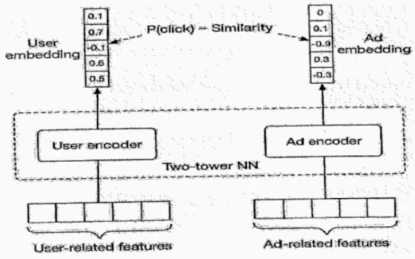
Cons:

- Cannot capture complex interactions - pairwise feature interaction.
- Continual learning is slow.

Model Development : Model Selection.
Neural Networks



1. Single NN



2. Two Tower NN.

Still not the best Choice because…

- Sparsity.
- Difficult to capture all pairwise feature interactions due to large number of features.

Deep & Cross Network (DCN):

- Deep Network - learns complex and generalizable features using deep neural network.
- Cross Network - Automatically captures feature interactions and learns good features.

The Cross Network in DCN explicitly models feature interactions to capture complex relationships in data, complementing the Deep Network's hierarchical feature learning.

Example Scenario:

In an e-commerce recommendation system, features like **User Age**, **Item Price**, and **Item Category** interact. For instance:

- **Younger users may prefer lower-priced items.**
- **Specific genders might favor certain categories.**

Cross Network Process:

Let x be the original feature vector. At each layer, the Cross Network computes:

$$x^{(l+1)} = x^{(l)} \cdot w + b + x$$

Where w (weights) captures interactions, and x (original input) ensures relationships are computed iteratively.

Benefits:

The network layers automatically learn interactions like:

- Younger females browsing low-priced fashion during sales.
- Middle-aged males browsing high-priced electronics.

By explicitly modeling interactions, the Cross Network avoids manual feature engineering, making it efficient for tasks like recommendations, especially when paired with the Deep Network for generalization.

Cross Network in DCN:

The **Cross Network** explicitly models interactions between features. For example, if you have features x_1 (user age) and x_2 (product price), a Cross Layer calculates new features like $x_1 \cdot x_2$, $x_1 \cdot x_3$, etc., stacking these computations across layers.

Each layer refines the feature interactions by adding context or depth:

1. **First Layer**: Learns basic pairwise interactions (e.g., age * price).
2. **Second Layer**: Builds on the first, learning interactions like "age * price * category."
3. **Further Layers**: Add higher-order interactions iteratively.

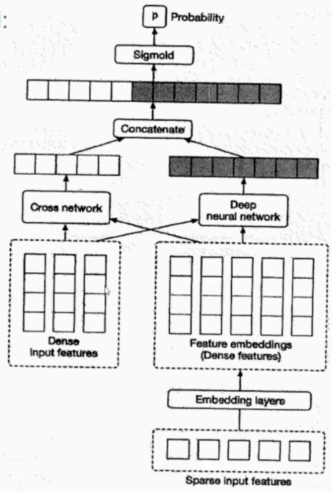
Feature	Two-Tower Network	Deep & Cross Network (DCN)
Architecture	Consists of two separate neural networks (towers): one for user features and one for item features. The outputs of these towers are then combined (e.g., using dot product or concatenation) to produce a relevance score.	Combines a "deep network" (DNN) with a "cross network." The DNN learns high-order feature interactions implicitly, while the cross network explicitly learns bounded-degree feature crosses.
Feature Interactions	Primarily relies on the combination of the final embeddings from each tower to capture interactions. Implicitly learns some interactions within each tower, but the main interaction happens at the end.	Explicitly models feature crosses using the cross network, which efficiently captures interactions between features. Also learns implicit interactions through the DNN.
Computational Cost	Generally more computationally efficient, especially for serving recommendations in real-time, as the user and item embeddings can be pre-computed and stored. The final combination step is relatively fast.	Can be computationally expensive, especially with a large number of features, as the cross network involves explicit feature crossing. However, it's more efficient than exhaustively computing all possible feature combinations.
Training	Trains the two towers separately or jointly. Can be trained using various loss functions, such as pairwise ranking loss or pointwise loss.	Trained end-to-end, typically using a pointwise loss function (e.g., log loss).
Strengths	Scalable for large datasets and real-time serving. Good for scenarios where user and item features can be effectively represented by separate embeddings. Simple to implement and understand.	Effectively captures explicit feature crosses, which are important for capturing certain types of relationships. Can achieve good performance when explicit feature interactions are crucial.
Weaknesses	May not capture complex, high-order feature interactions as effectively as models that explicitly model them. The final combination step might be a bottleneck for capturing fine-grained interactions.	Can be computationally intensive with many features. The cross network is limited to bounded-degree crosses, so it might not fully capture very complex interactions.
When to Use	*** Large-scale recommendation systems with billions of users and items.** *** Real-time serving scenarios where low latency is crucial.** *** Scenarios where user and item features can be effectively separated and represented by embeddings (e.g., collaborative filtering).**	*** Scenarios where explicit feature interactions are known to be important (e.g., click-through rate prediction, ad targeting).** *** When computational resources are sufficient and high accuracy is prioritized.** *** When there's a need to capture both low-order and high-order feature interactions.**

- **Two-Tower:** Imagine having separate profiles for users and items. The Two-Tower model learns "summaries" (embeddings) of these profiles and then compares them to see how well they match. This is fast and scalable.

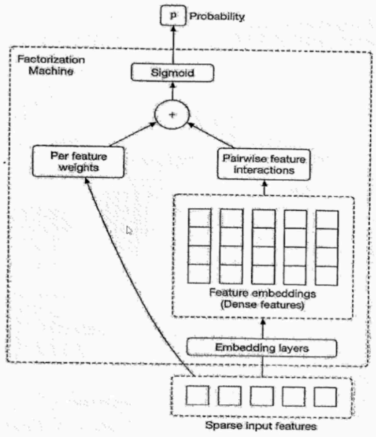
- **DCN:** Imagine having a detailed spreadsheet where you list all possible combinations of user and item features. DCN tries to learn which combinations are most important for making predictions. This is good for capturing specific relationships but can be more computationally intensive.

More reading on DCN: https://mlfrontiers.substack.com/p/the-rise-of-deep-and-cross-networks

Architecture of parallel DCN:



Model Development: Model Selection:
Factorization Machines: an embedding based model.



Sparse input features, typically one-hot encoded categorical variables, are fed to embedding layers using a technique called **embedding lookup**.

Here's how it works:

1. **Embedding Matrix:** An embedding layer contains an *embedding matrix*. This matrix has a size of *V x D*, where *V* is the vocabulary size (number of unique categories) and *D* is the embedding dimension (the size of the dense vector representation). Each row in the embedding matrix represents the embedding vector for a specific category.
2. **One-Hot Encoding as Indices:** The one-hot encoded input features are used as *indices* to look up the corresponding embeddings in the embedding matrix.
3. **Lookup Operation:** When a one-hot encoded vector is fed to the embedding layer, the index of the "1" is used to select the corresponding row from the embedding matrix. This selected row is the embedding vector for that category.

Let's say we have a categorical feature "City" with 5 possible values (New York, London, Tokyo, Paris, Sydney). We'll use an embedding dimension of 3.

- **One-hot encoding:**
  - New York: [1, 0, 0, 0, 0]
  - London: [0, 1, 0, 0, 0]
  - ...
- **Embedding Matrix (5x3):**

```
[[0.1, 0.2, 0.3],     // Embedding for New York
 [0.4, 0.5, 0.6],     // Embedding for London
 [0.7, 0.8, 0.9],     // Embedding for Tokyo
 [1.0, 1.1, 1.2],     // Embedding for Paris
 [1.3, 1.4, 1.5]]     // Embedding for Sydney
```

- **Lookup:** If the input is [0, 1, 0, 0, 0] (London), the embedding layer will look up the second row of the embedding matrix, which is [0.4, 0.5, 0.6]. This vector is the output of the embedding layer for this input.

Pairwise Interaction in FM

The pairwise interaction term is given by:

$$\text{Pairwise Interaction} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \cdot x_i \cdot x_j$$

Where:

- n: Number of features.
- x_i, x_j: Input feature values.
- $\mathbf{v}_i, \mathbf{v}_j$: Latent vectors (embeddings) of size k corresponding to features i and j.
- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$: Dot product of the latent vectors.

Why No Explicit Embeddings?

Even though you don't explicitly see embeddings in the input, **FM learns the latent vectors $\mathbf{v}_i$** for each feature during training. These vectors capture the importance of feature interactions.

Simplifying the Computation

To compute the pairwise interactions efficiently, FM uses this formula:

$$\frac{1}{2} \left(\left(\sum_{i=1}^{n} \mathbf{v}_i \cdot x_i \right)^2 - \sum_{i=1}^{n} \left(\mathbf{v}_i^2 \cdot x_i^2 \right) \right)$$

1. $\sum_{i=1}^{n} \mathbf{v}_i \cdot x_i$: Weighted sum of the latent vectors.
2. $(\cdot)^2$: Element-wise square to simulate interactions.
3. Subtract the squared terms to remove self-interactions.

Intuition Without Explicit Embedding

Even though you don't provide embeddings, the **model learns them during optimization**. The embeddings represent how features interact in a latent space, where the dot product measures their compatibility or interaction strength.

For example:

- If x_i = "Product A bought" and x_j = "Product B bought," the learned $\mathbf{v}_i$ and $\mathbf{v}_j$ might encode their compatibility (e.g., "products often bought together").

The interaction between two features is determined by dot product of their embeddings.

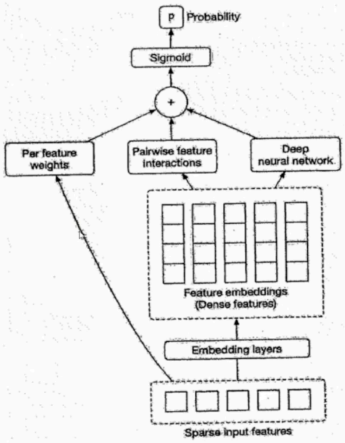
$$\hat{y}(x) = w_0 + \sum_i w_i x_i + \sum_i \sum_j \langle v_i, v_j \rangle x_i x_j$$

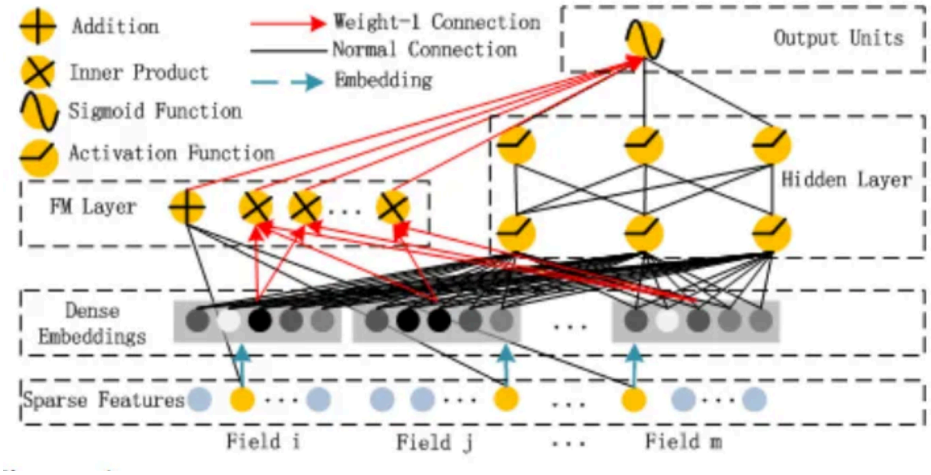
Logistic regression Pairwise interaction

- First 2 terms computes the linear combination of features, similar to Logistic Regression.
- Third term computes pairwise feature interaction.

More on FM: https://www.youtube.com/watch?v=rtKtZpXQSZg

Deep Factorization Machines (DeepFM):





Watch video on DeepFM: https://www.youtube.com/watch?v=w_YSl0Gd5CA

Read this: https://medium.com/data-science-in-your-pocket/deepfm-for-recommendation-systems-explained-with-codes-c200063990f7

Alternate way:

- Combining GBDT and DeepFM.

Model Training : Constructing the dataset.

- Input features are computed from the user and ad, depending on that the following labels will be assigned:
  - Positive label - if the user clicks on the ad in less than t seconds.
  - Negative label - if the user does not click the ad in less than t seconds.

#	User and interaction features	Ad features	Label
1	1 0 1 0.8 0.1 1 0	0 1 1 0.4 0.9 0	Positive
2	1 1 0 -0.6 0.9 1 1	1 1 0 0.2 0.7 1	Negative

- The model should be continuously trained in order to adapt new data. So new data points should be continuously generated using new interactions.
- We will us cross entropy as a classification loss function.

Evaluation Metrics : Offline Evaluation:
Two metrics typically used to evaluate for ad prediction system:

- Cross entropy
- Normalized cross-entropy (NCE)

- Cross entropy - in ideal system we have CE as 0 for negative classed and 1 for positive classes. Lower the CE higher the accuracy.of the prediction.

$$H(p,q) = -\sum_i p_i \log q_i = -\sum_i \left(y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i) \right)$$

Area Under the ROC Curve (AUC): This is a very common metric for binary classification tasks like click prediction. It measures the ability of the model to distinguish between positive and negative examples. A higher AUC is better (closer to 1).

Cross Entropy:

Cross-entropy measures the dissimilarity between two probability distributions: the true distribution (y) and the predicted distribution ($\hat{y}$). For a classification task:

$$\text{Cross Entropy} = -\sum_{i} y_i \log(\hat{y}_i)$$

- It penalizes predictions that deviate from the true labels.
- Values range from 0 (perfect prediction) to $+\infty$ (poor prediction).

Normalized Cross Entropy (NCE):

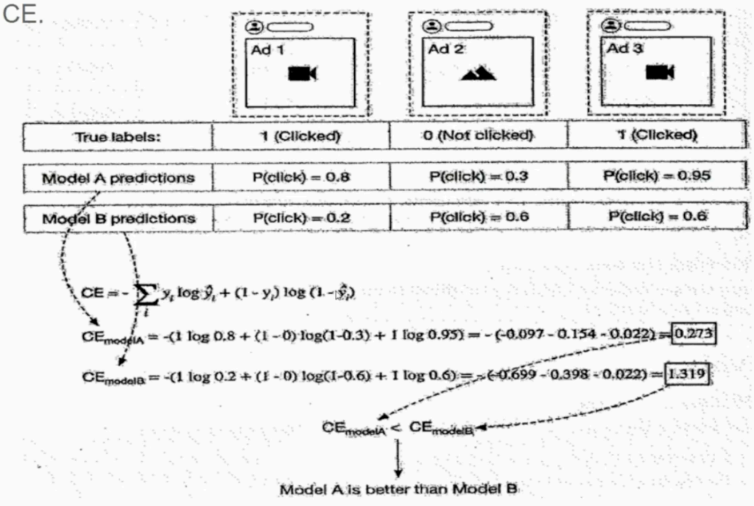
NCE scales the cross-entropy loss to a range of $[0, 1]$, making it interpretable regardless of dataset size or class imbalance:

$$\text{Normalized Cross Entropy (NCE)} = \frac{\text{Cross Entropy}}{\text{Maximum Possible Entropy}}$$

Key Differences:

1. **Interpretability:**
   - **Cross Entropy**: Absolute value, hard to compare across datasets.
   - **NCE**: Relative score between 0 (worst) and 1 (perfect).
2. **Purpose:**
   - **Cross Entropy**: Minimization during training.
   - **NCE**: Evaluation, comparison across datasets or models.
3. **Use Case:**
   - **Cross Entropy**: Direct loss in optimization.
   - **NCE**: Assess model performance ur[↓] varying data distributions.

Eg. for CE.



- Normalized Cross Entropy: is ratio of our model's CE and the CE of background average CTR of training data.

$$\text{Normalized cross entropy} = \frac{CE(\text{ML model})}{CE(\text{Simple baseline})}$$

- So, a low NCE indicates model outperforms the simple baseline and an NCE >= 1 shows that model is not performing better than simple baseline.

Online Metrics:

- CTR - number of clicked ads / number of ads shown
- Conversion rate - number of conversions / number of impressions
- Revenue Lift - percentage of revenue increase.
- Hide rate.- Number of ads hidden by user / number of show ads

Serving:

- Data Preparation Pipeline.
- Continual Learning Pipeline.
- Prediction Pipeline.