1. **Embedding Columns**:
   * When dealing with a large number of categories for a feature, one-hot encodings become infeasible.
   * Embedding columns represent categorical data as lower-dimensional dense vectors, allowing each cell to contain any number.
   * They are useful for finding nearest neighbors, input into a machine learning model, and visualization of concepts.
2. **Embeddings in Practice**:
   * Example: Handwritten digits dataset MNIST.
   * Embeddings help visualize data clusters and identify insights, such as variations in handwriting styles.
3. **Embeddings for Movie Recommendations**:
   * Problem: Recommending movies to users based on preferences.
   * Embeddings help organize movies by similarity using attributes like genre or popularity.
   * Adding dimensions (e.g., movie popularity) improves recommendations and better represents user interests.
4. **Hyperparameters for Embeddings**:
   * The number of embedding dimensions (d) is a hyperparameter set before training.
   * Higher dimensions allow for more accurate representation but increase the risk of overfitting and slower training.
   * A rule of thumb: Use the fourth root of the total number of possible values for the feature.
5. **Feature Crosses**:
   * Feature crosses combine features into synthetic features, enabling models to learn separate weights for each combination.
   * They represent non-linear relationships and are backed by hashed columns to handle large combinations.
6. **Training with Feature Columns**:
   * To train a model with feature columns, define an input function that returns features and labels.
   * Use the Keras API's **Model.fit()** function or a custom model for training.
   * Data should be passed as Numpy arrays or **tf.data.Dataset** objects.

These points provide insights into the use of embeddings, feature crosses, and training models with feature columns in machine learning applications, such as movie recommendations and real estate price prediction.

Here's a summary of the key points discussed regarding activation functions and their role in training deep neural network models:

1. **Linear Model Representation**:
   * A linear model involves combining input features with weights and bias terms to produce an output.
   * Adding hidden layers without non-linear activation functions results in the same linear model.
2. **Introducing Non-Linearity**:
   * Non-linear activation functions, such as sigmoid, tanh, or ReLU, introduce non-linearity to neural networks.
   * They act as transition points between layers, allowing the model to capture complex patterns and relationships.
3. **Role of Activation Functions**:
   * Activation functions prevent neural networks from collapsing into shallow models by introducing non-linear transformations.
   * Layers with linear activation functions can be collapsed into a single layer, while non-linear activation functions prevent this collapse.
4. **Common Activation Functions**:
   * Sigmoid and tanh were among the earliest activation functions but suffer from issues like saturation, leading to the vanishing gradient problem.
   * ReLU (Rectified Linear Unit) is a popular choice due to its simplicity and effectiveness, especially in speeding up training.
   * However, ReLU can suffer from "dying ReLU" problem where some neurons become inactive, leading to zero gradients and stalled training.
   * Various modifications to ReLU address its limitations, such as Leaky ReLU, Parametric ReLU, Exponential Linear Unit (ELU), and Gaussian Error Linear Unit (GELU).
5. **Visual Representation**:
   * Activation functions can be visualized on a graph to understand their behavior and characteristics.

By understanding the role and characteristics of different activation functions, practitioners can choose the most suitable one for their neural network architecture and optimize training efficiency and effectiveness.