00:00 - Neural net from scratch

04:46 - Parameters in PyTorch

07:42 - Embedding from scratch

12:21 - Embedding interpretation

VISUALISING BIAS:
What is the best way to understand and pick good movies?

Low user bias ->a user gives less ratings typically than an average user due to his preferences
[movie critic who is critical even about good movies]

High user bias -> A user gives high ratings typically than an average user
[movie lover user that likes even crappy movies]

High movie bias -> Movie seems to be liked typically by majority of users than expected
[Even people who don't normally like such movies like this movie]

Low movie bias -> Movie seems to be less liked typically by majority of users than expected
[Even the people who typically like such movies, don't like it much]

VISUALIZING EMBEDDINGS:
PCA -> Principal component analysis can be done on the latent factors and then see the most important factors and visualize this to understand the division of different movies

18:06 - Collab filtering in fastai

22:11 - Embedding distance

nn.CosineSimilarity is one way such as euclidean distance to assess what other movies are like a specific movie based on the latent factors.

24:00 - Bootstrapping with collaborative model

24:22 - Collab filtering with DL

Fastai also recommends the number of embeddings that we can use.
We can also concatenate metadata about users and movies to the respective embeddings.

30:25 - Embeddings for NLP

Embeddings are used in NLP a lot. Words and texts are inherently categorical and sequential. In a text from the training dataset, all the unique words are identified and are assigned numerical tokens. Now, each token is assigned an embedding vector of finite dimensions. These vectors are concatenated and the model is trained with these. After training the model, words that often come together or appear in a similar context would exist close to one another in an embedding vector space. This helps the model to pick the right word according to the context.

Properties of NLP Embeddings:

- **Semantic Relationships:** Capture synonyms, analogies, and hierarchical structures.
- **Contextual Understanding:** Understands word meaning based on surrounding text.
- **Transfer Learning:** Pre-trained embeddings can be fine-tuned for various NLP tasks like sentiment analysis, translation, and question answering.

34:56 - Embeddings for tabular

Similarly, embeddings can also be used for tabular analysis. In the previous lectures, we have seen kaggle challenges such as Bluebook for bulldozers and product price prediction. These challenges have both categorical and continuous variables. We have tried using Random forests in these challenges. However, the random forests or gradient boosted trees can also be used with embeddings to create a much better result. We can also use these embeddings in neural networks.

Categorical data with many unique values might be very high dimensional data and sparse. When we create a one-hot encoding with such categorical variables, it can be very high dimensional and we can see that the categories would not have any ordinal relationship within itself. Instead we can have embeddings for every single category in categorical variables and concatenate with the rest of the dataset, the embeddings are trained to capture internal relations within the categorical variable.

How Tabular Embeddings Work:

- **Initialization:** Assign each unique category a random vector of specified dimension.
- **Training:** Embeddings are learned jointly with the main task (e.g., prediction) by minimizing a loss function.
- **Optimization:** Through backpropagation, embeddings are adjusted to improve task performance, capturing relevant patterns.

Example Workflow:

1. **Input Data:** A dataset with categorical features like "country," "device type," etc.
2. **Embedding Layers:** Each categorical feature is passed through its own embedding layer.
3. **Concatenation:** Embeddings from all categorical features are concatenated with other features (numerical or processed text).
4. **Modeling:** The combined feature set is fed into subsequent layers (e.g., fully connected layers) for prediction tasks.
5. **Learning:** The entire network is trained end-to-end, updating embeddings to optimize performance.

Benefits of Tabular Embeddings:

- **Handling Rare Categories:** Learns meaningful representations even for infrequent categories by sharing latent features.
- **Interaction Modeling:** Facilitates learning complex interactions between different categorical features.
- **Flexibility:** Embeddings can adapt to various downstream tasks like classification, regression, or ranking

It is assumed that embedding layers are equivalent to extra layers on top of each one-hot encoded input. What does this mean?

- **Example:**
    - **Suppose you have three movies: `Movie A`, `Movie B`, and `Movie C`. Their one-hot encodings might be:**
        - **`Movie A`: [1, 0, 0]**
        - **`Movie B`: [0, 1, 0]**
        - **`Movie C`: [0, 0, 1]**

2. Embedding Layers:

- **What It Is: An embedding layer is a learnable mapping that transforms these one-hot encoded vectors into dense vectors of lower dimensionality.**
- **How It Works:**
    - **Instead of directly working with high-dimensional, sparse one-hot vectors, the embedding layer maps each category to a dense vector of a lower dimension, say `d`.**

    - **These dense vectors (embeddings) are learned during training and capture semantic or latent features of the categories.**

3. Equivalent to Extra Layers:

- **Understanding the Statement:**
- If you consider one-hot encoding, it produces a high-dimensional sparse vector. To get a dense representation, one could think of passing this one-hot vector through a linear layer (a matrix multiplication followed by an activation function).
- An embedding layer is conceptually equivalent to having this linear transformation directly on top of each one-hot encoded input, where the weights of this linear layer are the embedding matrix.
- Mathematically:
    - One-Hot Encoding to Dense Vector:
        - Given a one-hot encoded vector `x` of length `n`, and an embedding matrix `W` of shape `n x d`, where `d` is the embedding dimension.
        - The embedding layer operation can be seen as `x_embedded = W.T * x`, where `x_embedded` is the resulting dense vector of dimension `d`.
        - Here, `W.T` (the transpose of `W`) can be thought of as the weights of a linear layer that directly transforms the one-hot vector `x` into a dense vector.
- **Visualization:**
    - **One-Hot Vector: [0, 1, 0] (for `Movie B`)**

Embedding Matrix W (n x d):

```
W = | 0.2  0.8  0.1 |
    | 0.6  0.3  0.9 |
    | 0.4  0.7  0.5 |
```

- **Resulting Embedding:**
    - **For `Movie B`, the embedding is the second row of `W`: [0.6, 0.3, 0.9].**

44:33 - Convolutions

57:07 - Optimizing convolutions

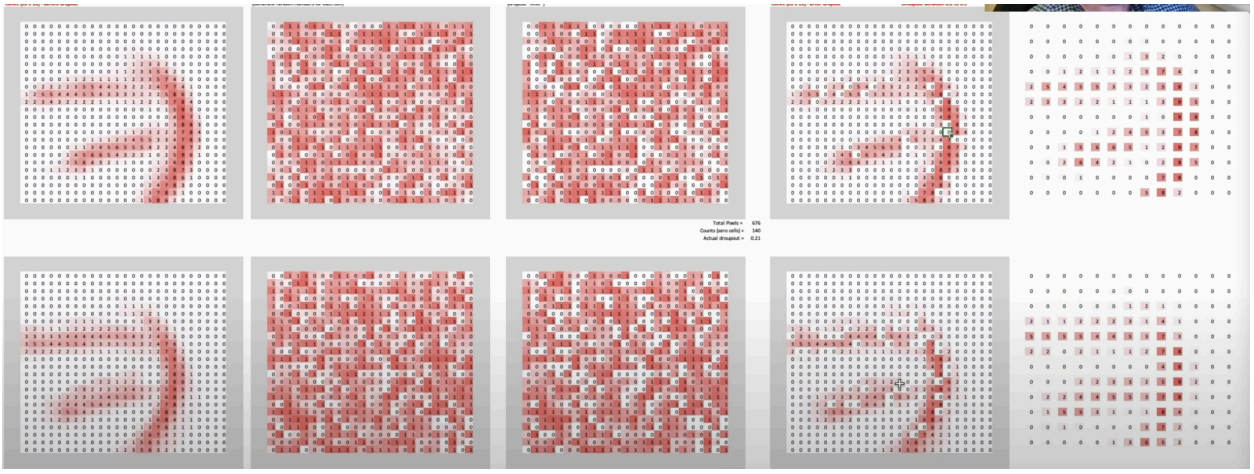
58:00 - Pooling

Nowadays instead of pooling, we are just using slide=2 in filters.
Insead of having a dense layer in the end, we keep doing the slide=2 until we get a small like 7x7 matrix and then do one single Average pooling -> this gives the average of all the activations(49 values). If we want to detect a small bear somewhere within the 49 values, we need to use max pooling. If one big bear -> average pooling.

1:05:12 - Convolutions as matrix products

1:08:21 - Dropout

We create a dropout mask for the activations and matrix multiply the mask with the original image. Essentially, we are randomly making some activations zero.



High dropout values indicate that a lot values would be set to zero and hence it would be very difficult to identify that the number is 7. The idea is that, when few values are missing in 7, we humans can interpolate it and can understand that the number is 7. We expect the computer to do the same. By using dropout, we can prevent the model from overfitting as dropping some values would make the model understand the inherent nature of 7 rather than just some surface level pixel relations.

1:14:27 - Activation functions