* **nn.Dropout**: When a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data. This "overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks and sets new records for speech and object recognition.
* The following line [getEmbeddings(\*o) for o in [(noUsers, n\_factors), (noMovies, n\_factors), (noUsers,1), (noMovies,1)]] initializes the embedding layers for the model. An embedding layer is a learnable lookup table that maps an integer index (in this case, the user ID and movie ID) to a dense vector of fixed size (in this case, n\_factors). The getEmbeddings function takes the number of categories (i.e. unique user and movie IDs) and the size of the embedding vectors as inputs, and returns a PyTorch nn.Embedding object that will be used as a layer in the neural network.
* The line [getEmbeddings(\*o) for o in [(noUsers, n\_factors), (noMovies, n\_factors), (noUsers,1), (noMovies,1)]] creates a list of tuples containing the number of categories (noUsers and noMovies) and the size of the embedding vectors (n\_factors and 1, respectively) for each embedding layer. The \*o syntax is used to unpack each tuple as separate arguments to the getEmbeddings function.