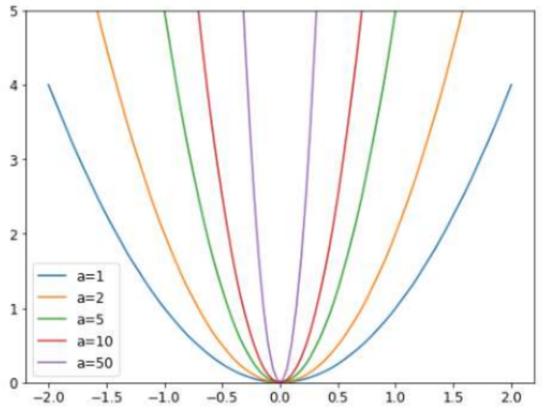
for previous notes
fast ai lesson 6 chpater 8 collborative filtering

weight decay :

```
    for preventing our model from overfitting we have to add the decay towards the model cause if it's making our model to look at broad view rather than looking for the
```

weight decay is trying to force the parameters unless they are really required to be big



the broder the a is more narrow the parabola will be
larger the a is narrower will be the parabola

when we decrease the weights we are possibly trying to avoid overfitting
larger a is we get 50 we have really narrow parabola big changes in the loss results in small changes to the parameters

`loss_with_weight_decay = loss +( wd * weights **2).sum()`

gradient of the `x**2` is just 2 times weight

weight decay referrs to adding on on the prameters
gives us the small flexibility

creating own embedding layer

embedding is just indexing in the array
normally a layer is created by inheriting Module (from pytorch)

to tell pytorch that something needs to be learned we have to wrap it with nn.parameter()

```
class T(Module):
        def __init__(self): self.a = nn.Parameter(torch.ones(3))

L(T().parameters())
```

here we are telling pytroch to rember torch.ones()
nn.Linear() si already calling nn.Parameter default so in this case we don't have to tell pytorch explicitely to remember the values

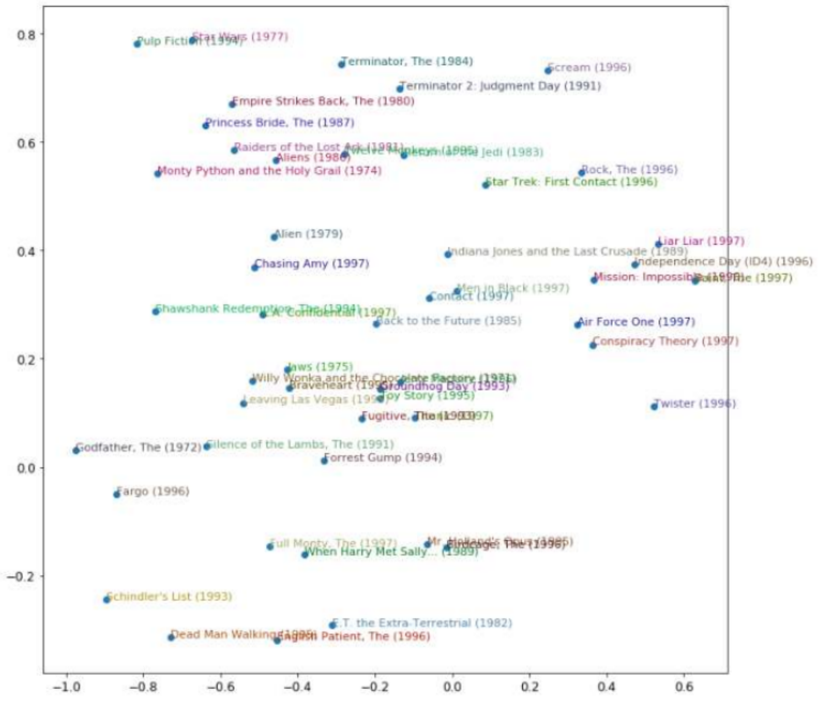
embeddings are just indexing of array to the values

```
#creating code embeddings
def create_params(size):
    return nn.Parameter(torch.zeros(*size).normal_(0,0.01))

class DotProductBias(Module):
    def __init__(self, n_users, n_movies, n_factors, y_range=(0,5.5)):
        self.user_factors = create_params([n_users,n_factors])
        self.user_bias = create_params([n_users])
        self.movie_factors = create_params([n_movies, n_factors])
        self.movie_bias = create_params([n_movies])
        self.y_range = y_range


    def forward(self, x ):
        users = self.user_factors[x[:,0]]
        movies = self.movie_factors[x[:,1]]
        res = (users*movies).sum(dim = 1)
        res += self.user_bias[x[:,0]] + self.movie_bias[x[:,1]]
        return sigmoid_range(res, *self.y_range)

# for fitting the model

model = DotProductBias(n_users,n_movies,50)
```

```
learn = Learner(dls, model , loss_func = MSELossFlat())
learn.fit_one_cycle(5, 5e-3,wd = 0.1)
```

interpreting what our model is doing in it's layers

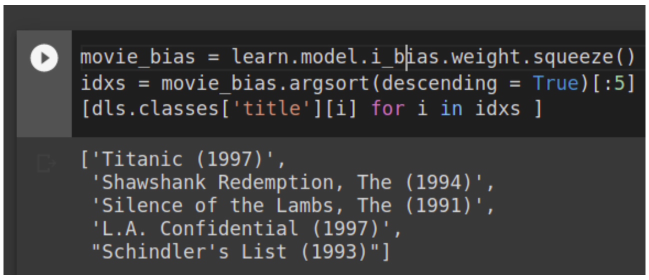
for doing the interpreatation we get the params through
learn.model.movie_bias.squeeze()
we can get indexing by using argsort over it
and we can found out which movies our model thinks are good and whch it should recommend based on the users who have watched movies before



Using fastai.collab

all the above steps which we did can be implemented by using fastai.collab.collab_learner in simple way

```
learn = collab_learner(dls, n_factors = 50 , y_range = (0,5.5))
learn.fit_one_cycle(5,5e-3, wd = 0.1)
learn.model # it'll give us model
```



```
movie_bias = learn.model.i_bias.weight.squeeze()
idxs = movie_bias.argsort(descending = True)[:5]
[dls.classes['title'][i] for i in idxs ]
```

```
['Titanic (1997)',
 'Shawshank Redemption, The (1994)',
 'Silence of the Lambs, The (1991)',
 'L.A. Confidential (1997)',
 "Schindler's List (1993)"]
```

here we are trying to get the top predictions from model

embedding distances

finding the distances from one movie to another movies in latent space

BootStraing problem:

In recommendation systems tehre is one problem when not much data of user is available consider when the user is new to the system which movies you will recommend you can consider the standard mean of the weights and recommend that movies but that can be biased
and one other thing is what if company is small and it doesn't have lot of users data available in that case building this type of recommendation model is not a good choice
even large companies like netflix ask you to select genere you like at the time of sign up so that it'll help them to recommend you some movies

since the smaller no of entuasist process can have large impact like you can take as anime genere the peopl ewho watch anime tends to watch them alot and having high no of ratings so here our model can become biased towards anime

Such a problem can change the entire makeup of your user base, and the behavior of your system. This is particularly true because of positive feedback loops. If a small number of your users tend to set the direction of your recommendation system, they are naturally going to end up attracting more people like them to your system. And that will, of course, amplify the original representation bias. This type of bias is a natural tendency to be amplified exponentially. You may have seen examples of company executives expressing surprise at how their online platforms rapidly deteriorated in

such a way that they expressed values at odds with the values of the founders. In the
presence of these kinds of feedback loops, it is easy to see how such divergence can happen quickly and in a way that it's hidden until it's too late

Deep learning for Collaborative Filtering :

since we are combining the embedding matrices rahter than taking the dot product the fast ai is having built in method for doing the same `get_emb_sz(dls)` which will return the recommended embedding sizes for the dataloader we pass

```
bs = get_emb_sz(dls)
embs
[(944, 74), (1635, 101)]
```

`*`kwargs and @deligates :

many of the libraries use kwargs to avoid the repeatatoin of the arguments cause they have inherited them from some other classes for showing the argumentes to user you can use @deligates before class declaration
since kwargs hide the params it's hard to read the documenatation and autocomplete doesn't usually work for the parameters which are there in kwargs

Furhtermore in upcoming 9th lesson we are gonna learn about handeling tabular data more generally