```
#hide
! [ -e /content ] && pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 719.8/719.8 KB 10.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 469.0/469.0 KB 37.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 55.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 68.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.2/212.2 KB 11.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 KB 10.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.8/199.8 KB 14.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 28.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.9/132.9 KB 6.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 45.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 264.6/264.6 KB 8.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.8/158.8 KB 9.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.2/114.2 KB 8.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 43.3 MB/s eta 0:00:00
Mounted at /content/gdrive
```

```
#hide
from fastbook import *
```

# Collaborative Filtering Deep Dive

## A First Look at the Data

```
from fastai.collab import *
from fastai.tabular.all import *
path = untar_data(URLs.ML_100k)
```

100.15% [4931584/4924029 00:00<00:00]

```
ratings = pd.read_csv(path/'u.data', delimiter='\t', header=None,
                      names=['user','movie','rating','timestamp'])
ratings.head()
```

| | user | movie | rating | timestamp |
|---|---|---|---|---|
| **0** | 196 | 242 | 3 | 881250949 |
| **1** | 186 | 302 | 3 | 891717742 |
| **2** | 22 | 377 | 1 | 878887116 |
| **3** | 244 | 51 | 2 | 880606923 |
| **4** | 166 | 346 | 1 | 886397596 |

```
last_skywalker = np.array([0.98,0.9,-0.9])
```

```
user1 = np.array([0.9,0.8,-0.6])
```

```
(user1*last_skywalker).sum()
```

```
2.1420000000000003
```

```
casablanca = np.array([-0.99,-0.3,0.8])
```

```
(user1*casablanca).sum()
```

```
-1.611
```

## Learning the Latent Factors

## Creating the DataLoaders

```
movies = pd.read_csv(path/'u.item',  delimiter='|', encoding='latin-1',
                     usecols=(0,1), names=('movie','title'), header=None)
movies.head()
```

| | movie | title |
|---|---|---|
| **0** | 1 | Toy Story (1995) |
| **1** | 2 | GoldenEye (1995) |
| **2** | 3 | Four Rooms (1995) |
| **3** | 4 | Get Shorty (1995) |
| **4** | 5 | Copycat (1995) |

```
ratings = ratings.merge(movies)
ratings.head()
```

| | user | movie | rating | timestamp | title |
|---|---|---|---|---|---|
| **0** | 196 | 242 | 3 | 881250949 | Kolya (1996) |
| **1** | 63 | 242 | 3 | 875747190 | Kolya (1996) |
| **2** | 226 | 242 | 5 | 883888671 | Kolya (1996) |
| **3** | 154 | 242 | 3 | 879138235 | Kolya (1996) |
| **4** | 306 | 242 | 5 | 876503793 | Kolya (1996) |

```
dls = CollabDataLoaders.from_df(ratings, item_name='title', bs=64)
dls.show_batch()
```

| | user | title | rating |
|---|---|---|---|
| **0** | 542 | My Left Foot (1989) | 4 |
| **1** | 422 | Event Horizon (1997) | 3 |
| **2** | 311 | African Queen, The (1951) | 4 |
| **3** | 595 | Face/Off (1997) | 4 |
| **4** | 617 | Evil Dead II (1987) | 1 |
| **5** | 158 | Jurassic Park (1993) | 5 |
| **6** | 836 | Chasing Amy (1997) | 3 |
| **7** | 474 | Emma (1996) | 3 |
| **8** | 466 | Jackie Chan's First Strike (1996) | 3 |
| **9** | 554 | Scream (1996) | 3 |

```
dls.classes
```

```
{'user': ['#na#', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145,
146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245,
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320,
321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345,
346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370,
371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395,
396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420,
421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445,
446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470,
471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495,
496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520,
521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570,
571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595,
596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620,
621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645,
646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670,
671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695,
696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720,
721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745,
746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770,
```

```
771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795,
796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820,
821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845,
846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870,
871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895,
896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920,
921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943],
 'title': ['#na#', "'Til There Was You (1997)", '1-900 (1994)', '101 Dalmatians (1996)', '12 Angry Men (1957)', '187 (1997)', '2
Days in the Valley (1996)', '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (1968)', '3 Ninjas: High Noon At Mega
Mountain (1998)', '39 Steps, The (1935)', '8 1/2 (1963)', '8 Heads in a Duffel Bag (1997)', '8 Seconds (1994)', 'A Chef in Love
(1996)', 'Above the Rim (1994)', 'Absolute Power (1997)', 'Abyss, The (1989)', 'Ace Ventura: Pet Detective (1994)', 'Ace
Ventura: When Nature Calls (1995)', 'Across the Sea of Time (1995)', 'Addams Family Values (1993)', 'Addicted to Love (1997)',
'Addiction, The (1995)', 'Adventures of Pinocchio, The (1996)', 'Adventures of Priscilla, Queen of the Desert, The (1994)',
'Adventures of Robin Hood, The (1938)', 'Affair to Remember, An (1957)', 'African Queen, The (1951)', 'Afterglow (1997)', 'Age
of Innocence, The (1993)', 'Aiqing wansui (1994)', 'Air Bud (1997)', 'Air Force One (1997)', 'Air Up There, The (1994)',
'Airheads (1994)', 'Akira (1988)', 'Aladdin (1992)', 'Aladdin and the King of Thieves (1996)', 'Alaska (1996)', 'Albino
Alligator (1996)', 'Alice in Wonderland (1951)', 'Alien (1979)', 'Alien 3 (1992)', 'Alien: Resurrection (1997)', 'Aliens
(1986)', 'All About Eve (1950)', 'All Dogs Go to Heaven 2 (1996)', 'All Over Me (1997)', 'All Things Fair (1996)', 'Alphaville
(1965)', 'Amadeus (1984)', 'Amateur (1994)', 'Amazing Panda Adventure, The (1995)', 'American Buffalo (1996)', 'American Dream
(1990)', 'American President, The (1995)', 'American Strays (1996)', 'American Werewolf in London, An (1981)', 'American in
Paris, An (1951)', 'Amistad (1997)', "Amityville 1992: It's About Time (1992)", 'Amityville 3-D (1983)', 'Amityville Curse, The
(1990)', 'Amityville Horror, The (1979)', 'Amityville II: The Possession (1982)', 'Amityville: A New Generation (1993)',
'Amityville: Dollhouse (1996)', 'Amos & Andrew (1993)', 'An Unforgettable Summer (1994)', 'Anaconda (1997)', 'Anastasia (1997)',
'Andre (1994)', 'Angel Baby (1995)', 'Angel and the Badman (1947)', 'Angel on My Shoulder (1946)', 'Angela (1995)', 'Angels and
Insects (1995)', 'Angels in the Outfield (1994)', 'Angus (1995)', 'Anna (1996)', 'Anna Karenina (1997)', 'Anne Frank Remembered
(1995)', 'Annie Hall (1977)', 'Another Stakeout (1993)', "Antonia's Line (1995)", 'Aparajito (1956)', 'Apartment, The (1960)',
'Apocalypse Now (1979)', 'Apollo 13 (1995)', 'Apostle, The (1997)', 'Apple Dumpling Gang, The (1975)', "April Fool's Day
(1986)", 'Apt Pupil (1998)', 'Aristocats, The (1970)', 'Army of Darkness (1993)', 'Around the World in 80 Days (1956)',
```

```
n_users  = len(dls.classes['user'])
n_movies = len(dls.classes['title'])
n_factors = 5

user_factors = torch.randn(n_users, n_factors)
movie_factors = torch.randn(n_movies, n_factors)
```

```
one_hot_3 = one_hot(3, n_users).float()
```

```
user_factors.t() @ one_hot_3
```

```
tensor([-0.4586, -0.9915, -0.4052, -0.3621, -0.5908])
```

```
user_factors[3]
```

```
tensor([-0.4586, -0.9915, -0.4052, -0.3621, -0.5908])
```

## Collaborative Filtering from Scratch

```
class Example:
    def __init__(self, a): self.a = a
    def say(self,x): return f'Hello {self.a}, {x}.'
```

```
ex = Example('Sylvain')
ex.say('nice to meet you')
```

```
'Hello Sylvain, nice to meet you.'
```

```
class DotProduct(Module):
    def __init__(self, n_users, n_movies, n_factors):
        self.user_factors = Embedding(n_users, n_factors)
        self.movie_factors = Embedding(n_movies, n_factors)

    def forward(self, x):
        users = self.user_factors(x[:,0])
        movies = self.movie_factors(x[:,1])
        return (users * movies).sum(dim=1)
```

```
x,y = dls.one_batch()
x.shape
```

```
torch.Size([64, 2])
```

```
model = DotProduct(n_users, n_movies, 50)
learn = Learner(dls, model, loss_func=MSELossFlat())
```

```
learn.fit_one_cycle(5, 5e-3)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 1.344786 | 1.279100 | 00:15 |
| 1 | 1.093331 | 1.109981 | 00:09 |
| 2 | 0.958258 | 0.990199 | 00:07 |
| 3 | 0.814234 | 0.894916 | 00:07 |
| 4 | 0.780714 | 0.882022 | 00:07 |

```
class DotProduct(Module):
    def __init__(self, n_users, n_movies, n_factors, y_range=(0,5.5)):
        self.user_factors = Embedding(n_users, n_factors)
        self.movie_factors = Embedding(n_movies, n_factors)
        self.y_range = y_range

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

```
model = DotProduct(n_users, n_movies, 50)
learn = Learner(dls, model, loss_func=MSELossFlat())
learn.fit_one_cycle(5, 5e-3)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.986799 | 1.005294 | 00:08 |
| 1 | 0.878134 | 0.918898 | 00:06 |
| 2 | 0.675850 | 0.875467 | 00:08 |
| 3 | 0.483372 | 0.877939 | 00:06 |
| 4 | 0.378927 | 0.881887 | 00:08 |

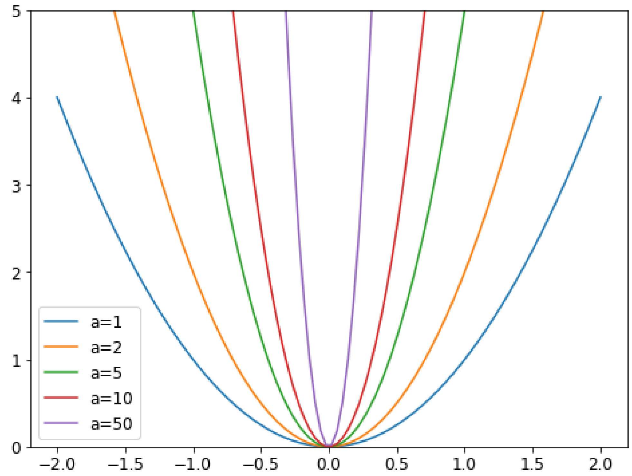
```
class DotProductBias(Module):
    def __init__(self, n_users, n_movies, n_factors, y_range=(0,5.5)):
        self.user_factors = Embedding(n_users, n_factors)
        self.user_bias = Embedding(n_users, 1)
        self.movie_factors = Embedding(n_movies, n_factors)
        self.movie_bias = Embedding(n_movies, 1)
        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, keepdim=True)
        res += self.user_bias(x[:,0]) + self.movie_bias(x[:,1])
        return sigmoid_range(res, *self.y_range)
```

```
model = DotProductBias(n_users, n_movies, 50)
learn = Learner(dls, model, loss_func=MSELossFlat())
learn.fit_one_cycle(5, 5e-3)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.938634 | 0.952516 | 00:08 |
| 1 | 0.846664 | 0.865633 | 00:09 |
| 2 | 0.608090 | 0.865127 | 00:08 |
| 3 | 0.413482 | 0.887318 | 00:08 |
| 4 | 0.286971 | 0.894876 | 00:08 |

## Weight Decay

```
x = np.linspace(-2,2,100)
a_s = [1,2,5,10,50]
ys = [a * x**2 for a in a_s]
_,ax = plt.subplots(figsize=(8,6))
for a,y in zip(a_s,ys): ax.plot(x,y, label=f'a={a}')
ax.set_ylim([0,5])
ax.legend();
```




```
model = DotProductBias(n_users, n_movies, 50)
learn = Learner(dls, model, loss_func=MSELossFlat())
learn.fit_one_cycle(5, 5e-3, wd=0.1)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.932776 | 0.961672 | 00:07 |
| 1 | 0.888625 | 0.882614 | 00:09 |
| 2 | 0.771066 | 0.832743 | 00:07 |
| 3 | 0.599807 | 0.822374 | 00:09 |
| 4 | 0.504981 | 0.822528 | 00:08 |

## Creating Our Own Embedding Module

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

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

```
(#0) []
```

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

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

```
(#1) [Parameter containing:
tensor([1., 1., 1.], requires_grad=True)]
```

```
class T(Module):
    def __init__(self): self.a = nn.Linear(1, 3, bias=False)

t = T()
L(t.parameters())
```

```
(#1) [Parameter containing:
tensor([[-0.3292],
        [-0.8623],
        [ 0.0592]], requires_grad=True)]
```

```
type(t.a.weight)
```

```
torch.nn.parameter.Parameter
```

```
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)
```

```
model = DotProductBias(n_users, n_movies, 50)
learn = Learner(dls, model, loss_func=MSELossFlat())
learn.fit_one_cycle(5, 5e-3, wd=0.1)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.929254 | 0.953444 | 00:09 |
| 1 | 0.865246 | 0.878304 | 00:08 |
| 2 | 0.720294 | 0.838921 | 00:08 |
| 3 | 0.582796 | 0.829129 | 00:09 |
| 4 | 0.474043 | 0.829031 | 00:07 |

## Interpreting Embeddings and Biases

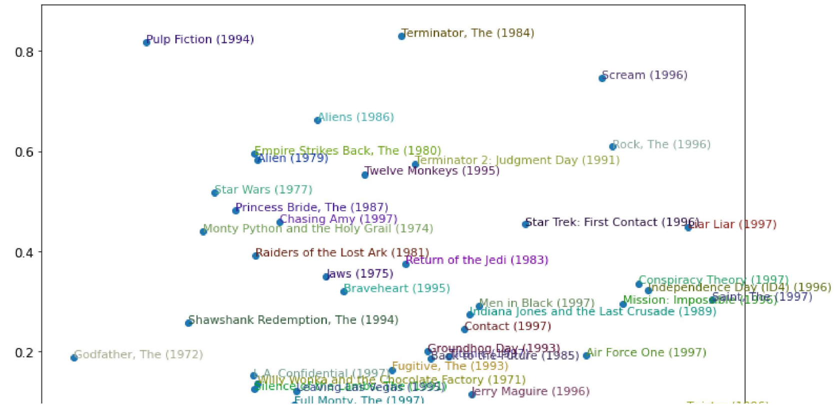
```
movie_bias = learn.model.movie_bias.squeeze()
idxs = movie_bias.argsort()[:5]
[dls.classes['title'][i] for i in idxs]
```

```
['Lawnmower Man 2: Beyond Cyberspace (1996)',
 'Children of the Corn: The Gathering (1996)',
 'Mortal Kombat: Annihilation (1997)',
 'Amityville 3-D (1983)',
 'Beautician and the Beast, The (1997)']
```

```
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)"]
```

```
g = ratings.groupby('title')['rating'].count()
top_movies = g.sort_values(ascending=False).index.values[:1000]
top_idxs = tensor([learn.dls.classes['title'].o2i[m] for m in top_movies])
movie_w = learn.model.movie_factors[top_idxs].cpu().detach()
movie_pca = movie_w.pca(3)
fac0,fac1,fac2 = movie_pca.t()
idxs = list(range(50))
X = fac0[idxs]
Y = fac2[idxs]
plt.figure(figsize=(12,12))
plt.scatter(X, Y)
for i, x, y in zip(top_movies[idxs], X, Y):
    plt.text(x,y,i, color=np.random.rand(3)*0.7, fontsize=11)
plt.show()
```




## Using fastai.collab

```
learn = collab_learner(dls, n_factors=50, y_range=(0, 5.5))
```

```
learn.fit_one_cycle(5, 5e-3, wd=0.1)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.939463 | 0.954959 | 00:09 |
| 1 | 0.841215 | 0.876151 | 00:08 |
| 2 | 0.724404 | 0.832099 | 00:08 |
| 3 | 0.597228 | 0.816953 | 00:09 |
| 4 | 0.481373 | 0.817286 | 00:07 |

```
learn.model
```

```
EmbeddingDotBias(
  (u_weight): Embedding(944, 50)
  (i_weight): Embedding(1665, 50)
  (u_bias): Embedding(944, 1)
  (i_bias): Embedding(1665, 1)
)
```

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

```
['L.A. Confidential (1997)',
 'Titanic (1997)',
 'Shawshank Redemption, The (1994)',
 'Silence of the Lambs, The (1991)',
 'Rear Window (1954)']
```

## Embedding Distance

```
movie_factors = learn.model.i_weight.weight
idx = dls.classes['title'].o2i['Silence of the Lambs, The (1991)']
distances = nn.CosineSimilarity(dim=1)(movie_factors, movie_factors[idx][None])
idx = distances.argsort(descending=True)[1]
dls.classes['title'][idx]
```

```
'Before the Rain (Pred dozhdot) (1994)'
```

## Bootstrapping a Collaborative Filtering Model

## Deep Learning for Collaborative Filtering

```
embs = get_emb_sz(dls)
embs
```

```
[(944, 74), (1665, 102)]
```

```
class CollabNN(Module):
    def __init__(self, user_sz, item_sz, y_range=(0,5.5), n_act=100):
        self.user_factors = Embedding(*user_sz)
        self.item_factors = Embedding(*item_sz)
        self.layers = nn.Sequential(
            nn.Linear(user_sz[1]+item_sz[1], n_act),
            nn.ReLU(),
            nn.Linear(n_act, 1))
        self.y_range = y_range

    def forward(self, x):
        embs = self.user_factors(x[:,0]),self.item_factors(x[:,1])
        x = self.layers(torch.cat(embs, dim=1))
        return sigmoid_range(x, *self.y_range)
```

```
model = CollabNN(*embs)
```

```
learn = Learner(dls, model, loss_func=MSELossFlat())
learn.fit_one_cycle(5, 5e-3, wd=0.01)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 0.943857 | 0.951898 | 00:11 |
| 1 | 0.914082 | 0.898525 | 00:09 |
| 2 | 0.848892 | 0.884356 | 00:08 |
| 3 | 0.814803 | 0.875278 | 00:09 |
| 4 | 0.761398 | 0.878594 | 00:08 |

```
learn = collab_learner(dls, use_nn=True, y_range=(0, 5.5), layers=[100,50])
learn.fit_one_cycle(5, 5e-3, wd=0.1)
```

| epoch | train_loss | valid_loss | time |
|---|---|---|---|
| 0 | 1.003178 | 0.998673 | 00:10 |
| 1 | 0.877362 | 0.934763 | 00:10 |
| 2 | 0.887651 | 0.898290 | 00:10 |
| 3 | 0.815599 | 0.865441 | 00:10 |
| 4 | 0.788975 | 0.864559 | 00:10 |

```
@delegates(TabularModel)
class EmbeddingNN(TabularModel):
    def __init__(self, emb_szs, layers, **kwargs):
        super().__init__(emb_szs, layers=layers, n_cont=0, out_sz=1, **kwargs)
```

### Sidebar: kwargs and Delegates

### End sidebar

## Conclusion

## Questionnaire

1. What problem does collaborative filtering solve?
2. How does it solve it?
3. Why might a collaborative filtering predictive model fail to be a very useful recommendation system?
4. What does a crosstab representation of collaborative filtering data look like?
5. Write the code to create a crosstab representation of the MovieLens data (you might need to do some web searching!).
6. What is a latent factor? Why is it "latent"?
7. What is a dot product? Calculate a dot product manually using pure Python with lists.
8. What does `pandas.DataFrame.merge` do?
9. What is an embedding matrix?
10. What is the relationship between an embedding and a matrix of one-hot-encoded vectors?

11. Why do we need `Embedding` if we could use one-hot-encoded vectors for the same thing?
12. What does an embedding contain before we start training (assuming we're not using a pretained model)?
13. Create a class (without peeking, if possible!) and use it.
14. What does `x[:,0]` return?
15. Rewrite the `DotProduct` class (without peeking, if possible!) and train a model with it.
16. What is a good loss function to use for MovieLens? Why?
17. What would happen if we used cross-entropy loss with MovieLens? How would we need to change the model?
18. What is the use of bias in a dot product model?
19. What is another name for weight decay?
20. Write the equation for weight decay (without peeking!).
21. Write the equation for the gradient of weight decay. Why does it help reduce weights?
22. Why does reducing weights lead to better generalization?
23. What does `argsort` do in PyTorch?
24. Does sorting the movie biases give the same result as averaging overall movie ratings by movie? Why/why not?
25. How do you print the names and details of the layers in a model?
26. What is the "bootstrapping problem" in collaborative filtering?
27. How could you deal with the bootstrapping problem for new users? For new movies?
28. How can feedback loops impact collaborative filtering systems?
29. When using a neural network in collaborative filtering, why can we have different numbers of factors for movies and users?
30. Why is there an `nn.Sequential` in the `CollabNN` model?
31. What kind of model should we use if we want to add metadata about users and items, or information such as date and time, to a collaborative filtering model?

## Further Research

1. Take a look at all the differences between the `Embedding` version of `DotProductBias` and the `create_params` version, and try to understand why each of those changes is required. If you're not sure, try reverting each change to see what happens. (NB: even the type of brackets used in `forward` has changed!)
2. Find three other areas where collaborative filtering is being used, and find out what the pros and cons of this approach are in those areas.
3. Complete this notebook using the full MovieLens dataset, and compare your results to online benchmarks. See if you can improve your accuracy. Look on the book's website and the fast.ai forum for ideas. Note that there are more columns in the full dataset—see if you can use those too (the next chapter might give you ideas).
4. Create a model for MovieLens that works with cross-entropy loss, and compare it to the model in this chapter.