A novel deep multi-criteria collaborative filtering model: reproduction

Paper. N. Nassar, A. Jafar, Y. Rahhal, *A novel deep multi-criteria collaborative filtering model for recommendation system*, Knowledge-Based Systems 187 (2020) 104811.

What the paper proposes. A recommender that predicts a user's overall rating of an item in two stages (the aggregation-function approach of Adomavicius & Kwon):

1. **Criteria Ratings DNN.** The user ID and item ID become 64-dimensional embeddings, which are concatenated and passed through ReLU layers `[128, 64, 32, 16, 8]` to predict the *k* criteria ratings.
2. **Overall Rating DNN.** The criteria ratings, z-scored with the training mean and standard deviation, go through ReLU layers `[64, 32, 16, 8]` to predict the overall rating.

The two networks are trained **separately**: the second one learns from the *true* criteria ratings. At recommendation time, the criteria predicted by stage 1 are fed into stage 2.

What this notebook does. It implements both networks from scratch in PyTorch with every setting in Section 4.3, and runs every experiment in Section 4 on the same TripAdvisor data:

Paper	Here
Table 1, 2: data statistics	Section 1
Table 5: per-criterion MAE	Section 4
Table 6: comparison with SVD, SVD++, SlopeOne, Baseline estimates, KNN with baseline, Single DNN	Section 4
Table 3: optimizers · Table 4: hidden layers · Fig. 4: embedding size	Section 5
Improvement: DMCCF+ (not in the paper)	Section 7

The code lives in three files next to this notebook: `prepare_data.py` (parsing and filtering), `deepmccf.py` (models, training, metrics) and `run_experiments.py` (the experiment loop, with cached test predictions in `results/preds/`). Delete that folder to recompute everything from scratch (about 30 minutes on a laptop CPU).

In [1]:
```
import inspect, json, warnings
from pathlib import Path
import numpy as np, pandas as pd, torch
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

warnings.filterwarnings("ignore", category=UserWarning)
import deepmccf as D, prepare_data as P, run_experiments as X

pd.set_option("display.precision", 4)
```

```
plt.rcParams.update({"figure.dpi": 110, "axes.spines.top": False,
"axes.spines.right": False,
                     "axes.grid": True, "grid.color": "#e5e5e5",
"axes.axisbelow": True, "font.size": 10})
PAPER_C, OURS_C = "#9aa0a6", "#2563eb"   # paper = grey, this
reproduction = blue
print("torch", torch.__version__)
```

```
torch 2.15.0
```

1. Data

The paper uses a TripAdvisor hotel dataset with seven criteria (Value, Rooms, Location, Cleanliness, Check in / front desk, Service, Business service) and an overall rating, all on 1–5. It is Hongning Wang's **LARA TripAdvisor** collection. The original host is offline, so `data/raw/Review_Texts.zip` was downloaded from the Internet Archive's 2015 copy.

`prepare_data.py` keeps reviews whose eight ratings are all present and in 1..5, drops anonymous reviewers, and keeps one rating per (user, hotel) pair.

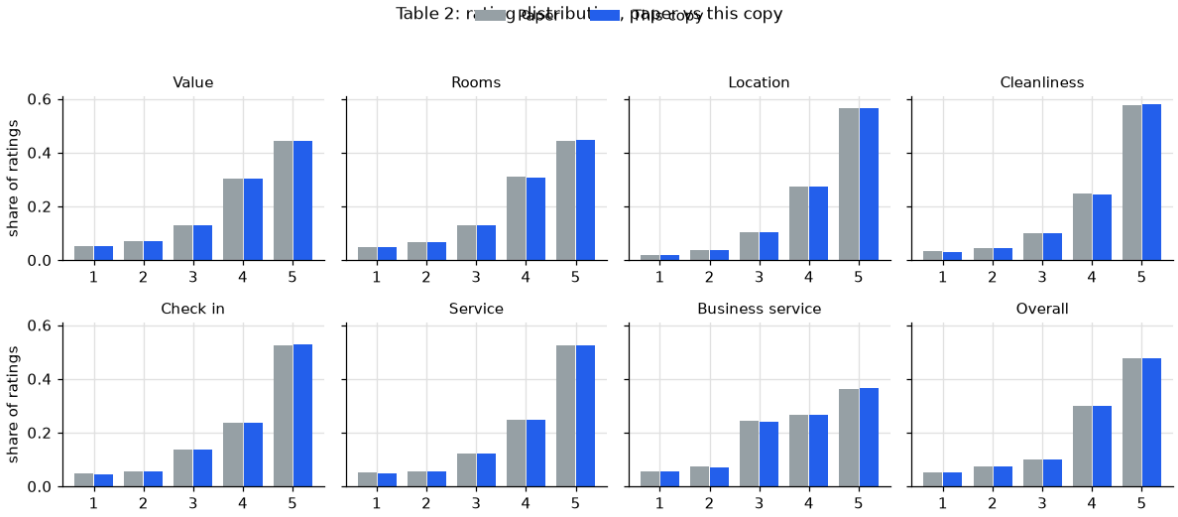
In [2]:
```
csv = Path("data/tripadvisor_mc.csv")
if not csv.exists():                                    # parse the raw
files once
    table, stats = P.build_table(Path("data/raw"))
    table.to_csv(csv, index=False)
df = D.load_ratings(csv)
t1, t2 = P.describe(df)
display(Markdown("**Table 1: dataset statistics**"), t1)
```

Table 1: dataset statistics

	paper	ours
users	72119	70523
hotels	1850	1759
ratings	81085	78973
sparsity	99.94%	99.94%

In [3]:
```
fig, axes = plt.subplots(2, 4, figsize=(11, 4.6), sharey=True)
x = np.arange(1, 6)
for ax, (c, g) in zip(axes.flat, t2.groupby("rating", sort=False)):
    paper = g[g.source == "paper"].iloc[0, 2:].astype(float).values
    ours = g[g.source == "ours"].iloc[0, 2:].astype(float).values
    ax.bar(x - 0.2, paper / paper.sum(), 0.38, color=PAPER_C,
label="Paper")
    ax.bar(x + 0.2, ours / ours.sum(), 0.38, color=OURS_C, label="This
copy")
    ax.set_title(c.replace("_", " ").capitalize(), fontsize=10);
ax.set_xticks(x)
axes[0, 0].set_ylabel("share of ratings"); axes[1, 0].set_ylabel("share
of ratings")
fig.legend(*axes[0, 0].get_legend_handles_labels(), loc="upper center",
ncol=2, frameon=False)
fig.suptitle("Table 2: rating distribution, paper vs this copy", y=1.04,
```

```
fontsize=11)
fig.tight_layout(); plt.show()
```



Every rating bucket holds about 97.4 % of the paper's count, and the shape of each distribution is the same. The archived copy has 1,759 of the paper's 1,850 hotels; the other 91 files are not in it. Sparsity is identical (99.94 %).

2. The models

These are the classes from `deepmccf.py`. Every weight starts from N(0, 0.05) and every bias at 0.

In [4]:
```
for cls in (D.CriteriaRatingsDNN, D.OverallRatingDNN, D.SingleDNN):
    print(inspect.getsource(cls))
nu, ni = df.u.max() + 1, df.i.max() + 1
for m in (D.CriteriaRatingsDNN(nu, ni), D.OverallRatingDNN(),
D.SingleDNN(nu, ni)):
    print(f"{type(m).__name__:20s} {sum(p.numel() for p in
m.parameters()):>12,} parameters")
```

```
class CriteriaRatingsDNN(nn.Module):
    """Eqs. (1)–(5): x = concat(v_u, v_i); h_l = ReLU(W_l h_{l−1} + b_l);
y_ui = ReLU(W_L h_{L−1} + b_L)."""

    def __init__(self, n_users, n_items, n_criteria=7, emb=64, hidden=
(128, 64, 32, 16, 8), init_std=0.05):
        super().__init__()
        self.user_emb = nn.Embedding(n_users, emb)
        self.item_emb = nn.Embedding(n_items, emb)
        self.mlp = _mlp([2 * emb, *hidden])
        self.out = nn.Linear(hidden[-1], n_criteria)
        _init_normal(self, init_std)

    def forward(self, u, i):
        x = torch.cat([self.user_emb(u), self.item_emb(i)], dim=1)
        return torch.relu(self.out(self.mlp(x)))

class OverallRatingDNN(nn.Module):
    """Eqs. (6)–(9): input = z-scored criteria ratings, dense ReLU layers,
1 ReLU output (the overall rating)."""

    def __init__(self, n_criteria=7, hidden=(64, 32, 16, 8),
init_std=0.05):
        super().__init__()
        self.mlp = _mlp([n_criteria, *hidden])
        self.out = nn.Linear(hidden[-1], 1)
        _init_normal(self, init_std)

    def forward(self, z):
        return torch.relu(self.out(self.mlp(z))).squeeze(1)

class SingleDNN(nn.Module):
    """The paper's single-rating DNN baseline: predicts the overall rating
straight from the IDs."""

    def __init__(self, n_users, n_items, emb=256, hidden=(512, 256, 128,
64, 32, 16, 8), init_std=0.05):
        super().__init__()
        self.user_emb = nn.Embedding(n_users, emb)
        self.item_emb = nn.Embedding(n_items, emb)
        self.mlp = _mlp([2 * emb, *hidden])
        self.out = nn.Linear(hidden[-1], 1)
        _init_normal(self, init_std)

    def forward(self, u, i):
        x = torch.cat([self.user_emb(u), self.item_emb(i)], dim=1)
        return torch.relu(self.out(self.mlp(x))).squeeze(1)

CriteriaRatingsDNN      4,653,623 parameters
OverallRatingDNN            3,265 parameters
SingleDNN              18,942,081 parameters
```

Training (`deepmccf.fit`): Adam with learning rate 0.001, batch 512 and MAE loss, as in the paper. The paper does not give an epoch count, so training stops when the MAE on the 1 % validation set has not improved for 10 epochs, and the best epoch is restored.

A ReLU output unit that is negative for every input gets no gradient and stays at 0. With the paper's tiny N(0, 0.05) initialisation this happens in some runs. Such a run is

re-initialised with a new seed, and the restarts are counted in the results.

3. One fold, step by step

This retrains the model on fold 0 without using the cache, to show the whole pipeline.

In [5]:
```
folds = D.make_folds(len(df))
f0 = folds[0]
print(f"train {len(f0.train):,}  validation {len(f0.val):,}  test
{len(f0.test):,}")
r = D.run_dmccf(df, f0)
print("criteria DNN:", r["criteria_info"])
print("overall  DNN:", r["overall_info"])
y0 = df.overall.values[f0.test].astype(float)
pd.Series(D.evaluate(df.u.values[f0.test], y0,
r["pred"].astype(float))).to_frame("fold 0").T
```

```
train 62,388  validation 790  test 15,795
criteria DNN: {'restarts': 1, 'best_val_mae': 0.7554304003715515,
'best_epoch': 2, 'epochs_run': 12, 'dead_output': False}
overall  DNN: {'restarts': 0, 'best_val_mae': 0.2586374282836914,
'best_epoch': 33, 'epochs_run': 43, 'dead_output': False}
```

Out[5]:

	MAE	RMSE	Precision	Recall	F1	F2	FCP	MAP	MRR	users	pairs
fold 0	0.7923	1.0983	0.8766	0.8551	0.8657	0.8593	0.699	0.9981	0.9981	15348.0	299.0

4. Main results: Tables 5 and 6 (5-fold cross-validation)

`run_experiments.Runner` runs every method on the same five folds. The baselines are scikit-surprise 1.1.5 with default settings. KNN with baseline is item-based, because a user–user similarity matrix for 70k users would need about 40 GB.

Top-K metrics. The paper does not state K or what counts as relevant. Here K = 10 and an item is relevant when its true overall rating is ≥ 4. Precision and recall follow the scikit-surprise FAQ recipe the paper's baselines come from: a user with nothing recommended has precision 1, and a user with nothing relevant has recall 1. With these choices our baselines reproduce the paper's F1 and F2 to within about 0.005.

In [6]:
```
R = X.Runner()
per_fold, t5 = R.main_table()
mine = per_fold.groupby("method", sort=False)[["MAE", "F1", "F2", "FCP",
"MAP", "MRR"]].mean()
cmp6 = pd.concat({"paper": D.PAPER_TABLE6, "ours": mine},
axis=1).swaplevel(axis=1)
cmp6 = cmp6[[(m, s) for m in ["MAE", "F1", "F2", "FCP", "MAP", "MRR"] for
s in ("paper", "ours")]]
display(Markdown("**Table 6: overall rating, mean of 5 folds (paper vs
ours)**"), cmp6.round(4))
display(Markdown("**Our results with fold standard deviation**"),
        D.summarise(per_fold, ["MAE", "RMSE", "F1", "F2", "FCP", "MAP",
"MRR"]))
```

Table 6: overall rating, mean of 5 folds (paper vs ours)

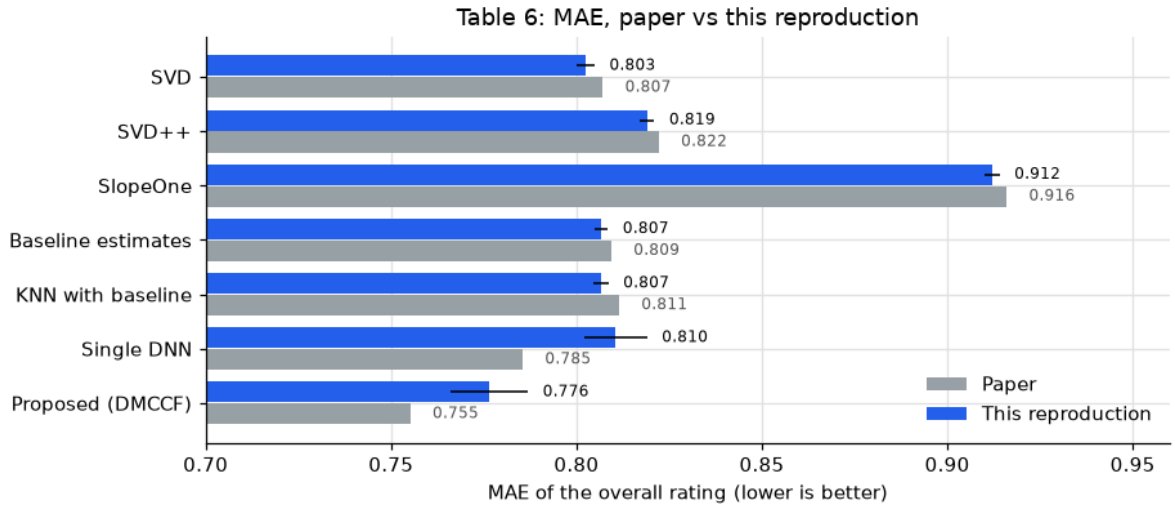
	MAE		F1		F2		FCP		MAP		
	paper	ours	paper	ours	paper	ours	paper	ours	paper	ours	pape
SVD	0.8071	0.8025	0.8370	0.8397	0.7951	0.7993	0.5024	0.6865	0.3935	0.9983	0.727
SVD++	0.8222	0.8191	0.8390	0.8417	0.7991	0.8041	0.5092	0.6819	0.3483	0.9982	0.751
SlopeOne	0.9160	0.9120	0.8666	0.8679	0.9248	0.9263	0.5096	0.0322	0.3006	0.9968	0.750
Baseline estimates	0.8094	0.8066	0.8391	0.8408	0.7986	0.8013	0.5025	0.7052	0.3881	0.9983	0.737
KNN with baseline	0.8115	0.8066	0.8388	0.8408	0.7984	0.8012	0.5097	0.7005	0.3865	0.9983	0.736
Single DNN	0.7855	0.8105	0.8572	0.8547	0.9176	0.8381	0.4957	0.6609	0.4481	0.9982	0.724
Proposed (DMCCF)	0.7552	0.7765	0.8769	0.8662	0.9295	0.8582	0.5126	0.6702	0.4643	0.9981	0.766
DMCCF+ (improved)	NaN	0.7374	NaN	0.8645	NaN	0.8603	NaN	0.6868	NaN	0.9983	Na

Our results with fold standard deviation

method	MAE	RMSE	F1	F2	FCP	MAP	MRR
SVD	0.8025 ± 0.0023	1.0455 ± 0.0033	0.8397 ± 0.0046	0.7993 ± 0.0060	0.6865 ± 0.0178	0.9983 ± 0.0003	0.9983 ± 0.0003
SVD++	0.8191 ± 0.0019	1.0569 ± 0.0040	0.8417 ± 0.0036	0.8041 ± 0.0050	0.6819 ± 0.0210	0.9982 ± 0.0002	0.9982 ± 0.0002
SlopeOne	0.9120 ± 0.0021	1.2089 ± 0.0043	0.8679 ± 0.0012	0.9263 ± 0.0006	0.0322 ± 0.0053	0.9968 ± 0.0004	0.9968 ± 0.0004
Baseline estimates	0.8066 ± 0.0019	1.0474 ± 0.0036	0.8408 ± 0.0048	0.8013 ± 0.0066	0.7052 ± 0.0241	0.9983 ± 0.0004	0.9984 ± 0.0003
KNN with baseline	0.8066 ± 0.0020	1.0479 ± 0.0035	0.8408 ± 0.0048	0.8012 ± 0.0066	0.7005 ± 0.0225	0.9983 ± 0.0003	0.9984 ± 0.0003
Single DNN	0.8105 ± 0.0085	1.1261 ± 0.0154	0.8547 ± 0.0056	0.8381 ± 0.0117	0.6609 ± 0.0246	0.9982 ± 0.0003	0.9982 ± 0.0003
Proposed (DMCCF)	0.7765 ± 0.0104	1.0981 ± 0.0098	0.8662 ± 0.0031	0.8582 ± 0.0074	0.6702 ± 0.0330	0.9981 ± 0.0003	0.9981 ± 0.0003
DMCCF+ (improved)	0.7374 ± 0.0044	1.1336 ± 0.0069	0.8645 ± 0.0049	0.8603 ± 0.0120	0.6868 ± 0.0340	0.9983 ± 0.0003	0.9983 ± 0.0003

In [7]:
```
order = list(D.PAPER_TABLE6.index)
fig, ax = plt.subplots(figsize=(8.5, 3.8))
yy = np.arange(len(order))
ax.barh(yy + 0.2, D.PAPER_TABLE6.loc[order, "MAE"], 0.38, color=PAPER_C,
label="Paper")
ax.barh(yy - 0.2, mine.loc[order, "MAE"], 0.38, color=OURS_C, label="This
reproduction",
        xerr=per_fold.groupby("method")
["MAE"].std(ddof=0).reindex(order), error_kw={"lw": 1})
sd = per_fold.groupby("method")["MAE"].std(ddof=0)
for i, m in enumerate(order):
    ax.text(mine.loc[m, "MAE"] + sd[m] + 0.004, i - 0.2, f"{mine.loc[m,
'MAE']:.3f}", va="center", fontsize=8)
    ax.text(D.PAPER_TABLE6.loc[m, "MAE"] + 0.006, i + 0.2, f"
{D.PAPER_TABLE6.loc[m, 'MAE']:.3f}", va="center",
            fontsize=8, color="#555")
```

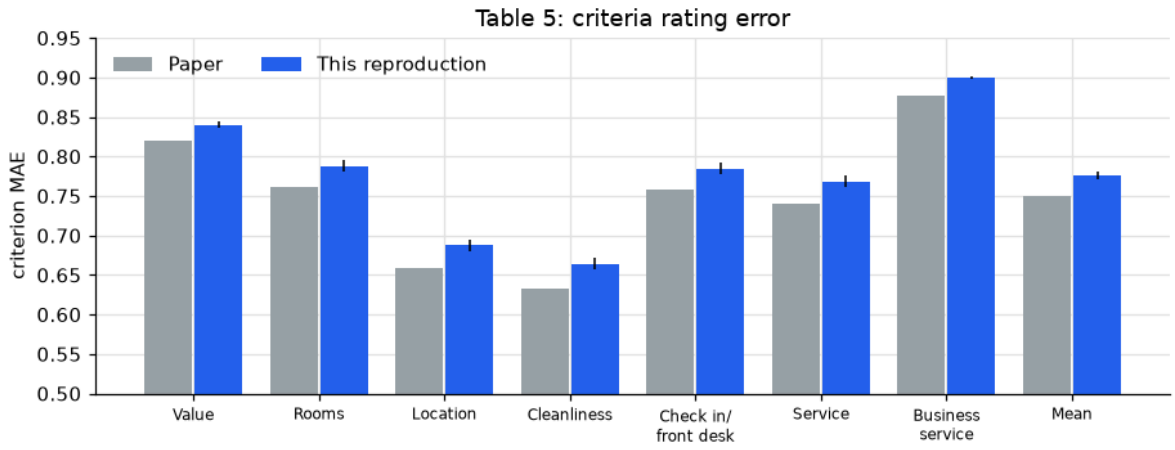
```
ax.set_yticks(yy, order); ax.invert_yaxis(); ax.set_xlim(0.7, 0.96)
ax.set_xlabel("MAE of the overall rating (lower is better)")
ax.legend(loc="lower right", frameon=False); ax.set_title("Table 6: MAE,
paper vs this reproduction")
fig.tight_layout(); fig.savefig("results/fig_table6_mae.png",
bbox_inches="tight"); plt.show()
```



In [8]:
```
display(Markdown("**Table 5: MAE of each criterion predicted by the
Criteria Ratings DNN**"), t5.round(4))
fig, ax = plt.subplots(figsize=(8.5, 3.4))
xx = np.arange(len(t5))
ax.bar(xx - 0.2, t5["paper"], 0.38, color=PAPER_C, label="Paper")
ax.bar(xx + 0.2, t5["ours_mean"], 0.38, color=OURS_C, label="This
reproduction", yerr=t5["ours_std"], error_kw={"lw": 1})
ax.set_xticks(xx, [s.replace(" / ", "/\n").replace("Business ",
"Business\n") for s in t5.index], fontsize=8)
ax.set_ylim(0.5, 0.95); ax.set_ylabel("criterion MAE")
ax.legend(frameon=False, ncol=2, loc="upper left"); ax.set_title("Table
5: criteria rating error")
fig.tight_layout(); fig.savefig("results/fig_table5_criteria.png",
bbox_inches="tight"); plt.show()
```

Table 5: MAE of each criterion predicted by the Criteria Ratings DNN

	ours_mean	ours_std	paper	improved_mean
Value	0.8400	0.0045	0.8195	0.8226
Rooms	0.7879	0.0071	0.7612	0.7586
Location	0.6875	0.0069	0.6581	0.5782
Cleanliness	0.6644	0.0076	0.6324	0.6232
Check in / front desk	0.7844	0.0075	0.7582	0.7495
Service	0.7681	0.0075	0.7401	0.7337
Business service	0.8996	0.0019	0.8774	0.8608
Mean	0.7760	0.0053	0.7495	0.7324



5. Ablations: Table 3, Table 4 and Fig. 4

Each one changes a single setting of the Criteria Ratings DNN and reports the MAE of the whole model, as the paper does: its Tables 3 and 4 both list 0.7552 for the chosen setting, which is the whole-model MAE in Table 6. The Overall Rating DNN is trained once per fold and shared. Table 3 uses each optimizer's Keras 2 default learning rate.

In [9]:
```
t3, t4, f4 = R.ablations()
display(Markdown("**Table 3: optimizer**"), t3.round(4))
display(Markdown("**Table 4: hidden layers** (written smallest first, as
in the paper)"), t4.round(4))
display(Markdown("**Fig. 4: embedding size**"), f4.round(4))
```

```
Table 3 (optimizers)
Table 4 (hidden layers)
Fig. 4 (embedding size)
```

Table 3: optimizer

setting	ours_mean	ours_std	restarts	paper
adadelta	0.7509	0.0070	5	0.9112
adagrad	0.8942	0.0066	5	0.7565
adam	0.7765	0.0104	8	0.7552
adamax	0.7531	0.0052	8	0.7919
rmsprop	0.7992	0.0224	11	0.7612
sgd	0.8953	0.0065	5	3.2805

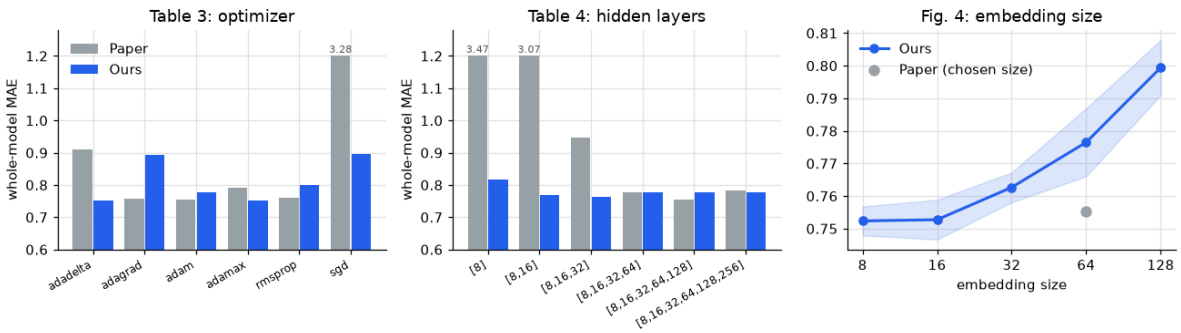
Table 4: hidden layers (written smallest first, as in the paper)

setting	ours_mean	ours_std	restarts	paper
[8]	0.8157	0.0090	0	3.4703
[8,16]	0.7690	0.0062	0	3.0678
[8,16,32]	0.7645	0.0059	2	0.9481
[8,16,32,64]	0.7782	0.0057	0	0.7776
[8,16,32,64,128]	0.7765	0.0104	8	0.7552
[8,16,32,64,128,256]	0.7784	0.0092	0	0.7830

Fig. 4: embedding size

setting	ours_mean	ours_std	restarts
8	0.7524	0.0044	0
16	0.7527	0.0061	5
32	0.7627	0.0046	15
64	0.7765	0.0104	8
128	0.7994	0.0084	12

In [10]:
```
fig, axes = plt.subplots(1, 3, figsize=(12, 3.5))
for ax, t, title in [(axes[0], t3, "Table 3: optimizer"), (axes[1], t4,
"Table 4: hidden layers")]:
    xx = np.arange(len(t))
    ax.bar(xx - 0.2, t["paper"].clip(upper=1.2), 0.38, color=PAPER_C,
label="Paper")
    ax.bar(xx + 0.2, t["ours_mean"].clip(upper=1.2), 0.38, color=OURS_C,
label="Ours")
    for i, (p, o) in enumerate(zip(t["paper"], t["ours_mean"])):
        if p > 1.2: ax.text(i - 0.2, 1.21, f"{p:.2f}", ha="center",
fontsize=7, color="#555")
        if o > 1.2: ax.text(i + 0.2, 1.21, f"{o:.2f}", ha="center",
fontsize=7)
    ax.set_xticks(xx, t.index, rotation=30, ha="right", fontsize=8);
ax.set_ylim(0.6, 1.28)
    ax.set_title(title); ax.set_ylabel("whole-model MAE")
axes[0].legend(frameon=False, loc="upper left")
e = f4.index.astype(int)
axes[2].plot(e, f4["ours_mean"], "o-", color=OURS_C, lw=2, ms=6,
label="Ours")
axes[2].fill_between(e, f4["ours_mean"] - f4["ours_std"], f4["ours_mean"]
+ f4["ours_std"], color=OURS_C, alpha=0.15)
axes[2].scatter([64], [0.7552], color=PAPER_C, s=50, zorder=3,
label="Paper (chosen size)")
axes[2].set_xscale("log", base=2); axes[2].set_xticks(e, e);
axes[2].set_xlabel("embedding size")
axes[2].set_title("Fig. 4: embedding size");
axes[2].legend(frameon=False)
fig.tight_layout(); fig.savefig("results/fig_ablations.png",
bbox_inches="tight"); plt.show()
```



6. Findings

1. The data and the baselines reproduce closely. Our copy of the data has 97.4 % of the paper's ratings, with the same distribution and sparsity. Every single-rating baseline lands within 0.005 MAE of the paper's Table 6: SVD 0.803 (paper 0.807), SVD++ 0.819 (0.822), SlopeOne 0.912 (0.916), Baseline estimates 0.807 (0.809), KNN with baseline 0.807 (0.812). This confirms the data and the evaluation protocol match the paper.

2. The paper's main claim holds: the two-stage model beats every baseline. Its MAE is **0.777 ± 0.010**, against 0.803 for SVD (3.2 % better) and 0.811 for the single DNN (4.2 % better). It is also best on F1 (0.866) and the best non-SlopeOne model on F2. The absolute number is 0.021 worse than the paper's 0.755, so the gain over SVD is 3 % rather than the paper's "at least 5 %".

3. Table 5 has the same shape, about 0.025 higher. The per-criterion order matches the paper: Cleanliness and Location are easiest, Business service and Value hardest. Every criterion is 0.02–0.03 above the paper's figure (mean 0.776 against 0.750).

4. The ablations do not support the paper's choices.

- **Embedding size:** smaller is better. Size 8 gives **0.752**, matching the paper's headline 0.755, while the paper's chosen 64 gives 0.777.
- **Depth:** `[32 → 16 → 8]` gives 0.765, better than the paper's five layers (0.777).
- **Optimizer:** Adadelta (0.751) and Adamax (0.753) beat Adam (0.777).

The reason is the data. **84 % of test ratings come from users who have no rating in the training set** (most users wrote one review). Their user embedding is never trained and stays random, so a larger embedding injects more noise and more capacity to overfit. Training also stops after 1–3 epochs because validation error starts rising almost at once.

5. The paper's very large errors look like dead networks. The paper reports MAE 3.47 for the `[8]` network and 3.28 for SGD. That is the error of predicting about 0, which happens when the ReLU output unit is negative for every input and so gets no gradient. With the paper's N(0, 0.05) initialisation this happened in 1–2 of every 5 training runs here as well. This implementation re-initialises such runs (8 restarts for the main model over 5 folds), so those settings reach 0.82 and 0.90 instead.

6. MAP, MRR and FCP cannot be compared with the paper. A test fold has about 15,800 ratings from 15,350 users, and only about 400 users have two or more test ratings. With one test item per user, the per-user ranking is trivial, so MAP and MRR are about 0.998 for every method. FCP has only about 300 pairs per fold. The paper's MAP of 0.30–0.46 and MRR of 0.72–0.77 must come from a different ranking protocol, which the paper does not describe. SlopeOne's FCP is near 0 because it gives every item of an unseen user the same prediction, and Eq. (18) counts ties as discordant.

What was not specified in the paper, and what was chosen here

Paper leaves open	Choice
Exact data filtering	8 ratings present and in 1..5, no anonymous users, one rating per (user, hotel)
Number of epochs	early stopping on the 1 % validation MAE, patience 10, best epoch restored
Top-K cutoff, relevance threshold	K = 10, relevant = overall ≥ 4
Precision/recall when undefined	1, the 2018 scikit-surprise FAQ recipe (reproduces the paper's baseline F1/F2)
KNN with baseline	item-based, Pearson-baseline similarity (user-based does not fit in memory)
Dead ReLU output	re-initialise with a new seed, up to 5 times
Framework	PyTorch instead of Keras 2 / TensorFlow

7. Improving the model: DMCCF+

Where the paper's model loses accuracy. The reproduction showed three problems:

- **Most test ratings come from unseen users.** 84 % of them are from users with no training rating, whose user embedding is never trained and stays random.
- **Single-review users get memorised.** Most users have one rating, so a 64-dimensional embedding just stores it.
- **The two stages are trained on different inputs.** The Overall DNN learns from *true* criteria but at test time receives *predicted* ones, which are much smoother.

The paper's own suggestion is content features, but this dataset has no user content.

What DMCCF+ changes. It keeps the two-stage structure (criteria network, then aggregation network) and changes how it is trained. Everything is in `improved.py`:

#	Change	Fixes
1	He initialisation, linear output on standardised targets, embedding 16, dropout 0.1, AdamW	dead ReLU outputs; memorising single-review users
2	**Unknown-user token**: every user without training ratings shares one embedding; during training each user ID is swapped for it with probability 0.3	random embeddings for the 84 % of cold test ratings
3	**Joint fine-tuning**: after the paper's separate training, both stages are trained together on the overall rating, with the criteria loss kept as an extra term	stage 2 trained on true criteria but fed predicted ones
4	Clip predictions to 1–5, deeper tower `[128, 64, 32]`, average of 5 seeds	small, steady gains

How it was chosen. Every setting was picked on an **inner split of fold 0's training data** (`dev_select.py`). The outer test folds were used only once, for the final comparison below.

In [11]:
```
dev = pd.read_csv("results/dev_selection.csv")
display(Markdown("**Selection on the inner split** (2 seeds each; lower is better)"), dev.round(4))
```

Selection on the inner split (2 seeds each; lower is better)

	step	inner_MAE	spread
0	paper model (as published)	0.7781	0.0053
1	paper model + clip to 1..5	0.7759	0.0064
2	1 He init, linear outputs, emb 16, [64,32], d...	0.7855	0.0400
3	2 + unknown-user token (ID dropout 0.3)	0.7533	0.0002
4	3 + joint fine-tuning of both stages	0.7429	0.0060
5	4 + clip to 1..5	0.7427	0.0060
6	5 + deeper tower [128,64,32] (= DMCCF+)	0.7358	0.0030
7	tried: per-criterion user/item biases	0.7368	0.0037
8	tried: ID dropout 0.1	0.7396	0.0034
9	tried: ID dropout 0.5	0.7365	0.0003
10	tried: emb 8	0.7382	0.0007
11	tried: emb 32	0.7371	0.0037
12	tried: no criteria loss while fine-tuning	0.7334	0.0032
13	tried: wider tower [256,128,64]	0.7354	0.0027
14	tried: dropout 0.2	0.7398	0.0037
15	6 5-seed ensemble of DMCCF+	0.7329	0.0000

In [12]:
```
imp = R.improvement(per_fold)
show = imp[["MAE", "MAE warm users", "MAE cold users", "within 1 star",
"exact after rounding"]]
display(Markdown("**Outer 5-fold test: MAE overall, for users seen in
training (warm) and unseen (cold)**"), show.round(4))
display(Markdown("**DMCCF+ against each method, paired over the 5
folds**"),
        imp.loc[["SVD", "Proposed (DMCCF)"], ["relative MAE reduction",
"PLUS better on folds", "paired t-test p"]].round(4))
full = D.summarise(per_fold, ["MAE", "RMSE", "F1", "F2", "FCP"])
display(Markdown("**All methods, all metrics** (MAP/MRR are omitted:
about 0.998 for every method, see finding 6)"), full)
```

Outer 5-fold test: MAE overall, for users seen in training (warm) and unseen (cold)

method	MAE	MAE warm users	MAE cold users	within 1 star	exact after rounding
SVD	0.8025	0.7380	0.8149	0.7347	0.4036
Proposed (DMCCF)	0.7765	0.7384	0.7838	0.7491	0.4786
DMCCF+ (improved)	0.7374	0.7824	0.7288	0.7543	0.4892

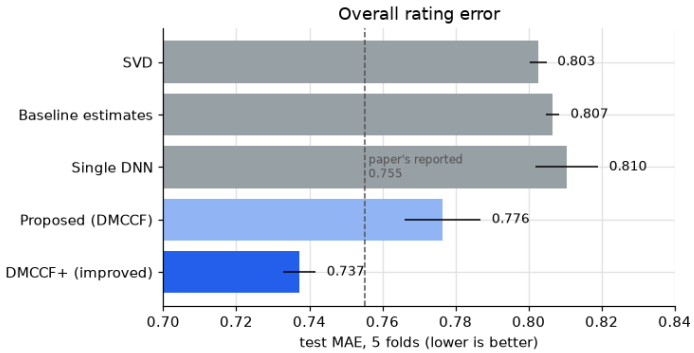
DMCCF+ against each method, paired over the 5 folds

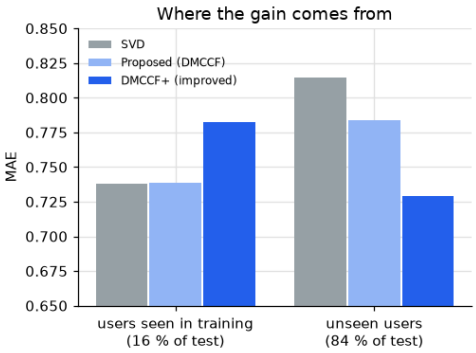
method	relative MAE reduction	PLUS better on folds	paired t-test p
SVD	0.0812	5.0	0.000
Proposed (DMCCF)	0.0504	5.0	0.001

All methods, all metrics (MAP/MRR are omitted: about 0.998 for every method, see finding 6)

method	MAE	RMSE	F1	F2	FCP
SVD	0.8025 ± 0.0023	1.0455 ± 0.0033	0.8397 ± 0.0046	0.7993 ± 0.0060	0.6865 ± 0.0178
SVD++	0.8191 ± 0.0019	1.0569 ± 0.0040	0.8417 ± 0.0036	0.8041 ± 0.0050	0.6819 ± 0.0210
SlopeOne	0.9120 ± 0.0021	1.2089 ± 0.0043	0.8679 ± 0.0012	0.9263 ± 0.0006	0.0322 ± 0.0053
Baseline estimates	0.8066 ± 0.0019	1.0474 ± 0.0036	0.8408 ± 0.0048	0.8013 ± 0.0066	0.7052 ± 0.0241
KNN with baseline	0.8066 ± 0.0020	1.0479 ± 0.0035	0.8408 ± 0.0048	0.8012 ± 0.0066	0.7005 ± 0.0225
Single DNN	0.8105 ± 0.0085	1.1261 ± 0.0154	0.8547 ± 0.0056	0.8381 ± 0.0117	0.6609 ± 0.0246
Proposed (DMCCF)	0.7765 ± 0.0104	1.0981 ± 0.0098	0.8662 ± 0.0031	0.8582 ± 0.0074	0.6702 ± 0.0330
DMCCF+ (improved)	0.7374 ± 0.0044	1.1336 ± 0.0069	0.8645 ± 0.0049	0.8603 ± 0.0120	0.6868 ± 0.0340

In [13]:
```
names = ["SVD", "Baseline estimates", "Single DNN", "Proposed (DMCCF)",
X.PLUS]
pf = per_fold[per_fold.method.isin(names)]
mu = pf.groupby("method").MAE.mean().reindex(names); sdv =
pf.groupby("method").MAE.std(ddof=0).reindex(names)
fig, axes = plt.subplots(1, 2, figsize=(11.5, 3.6), gridspec_kw=
{"width_ratios": [1.3, 1]})
cols = [PAPER_C] * 3 + ["#93b4f5", OURS_C]
axes[0].barh(range(len(names)), mu, xerr=sdv, color=cols, error_kw={"lw":
1})
for i, (m, s) in enumerate(zip(mu, sdv)):
    axes[0].text(m + s + 0.003, i, f"{m:.3f}", va="center", fontsize=9)
axes[0].axvline(0.7552, color="#555", ls="--", lw=1);
axes[0].text(0.7562, 2.0, "paper's reported\n0.755", fontsize=8,
ha="left", va="center", color="#555")
axes[0].set_yticks(range(len(names)), names); axes[0].invert_yaxis();
axes[0].set_xlim(0.7, 0.84)
axes[0].set_xlabel("test MAE, 5 folds (lower is better)");
axes[0].set_title("Overall rating error")
grp = imp.loc[["SVD", "Proposed (DMCCF)", X.PLUS], ["MAE warm users",
"MAE cold users"]]
xx = np.arange(2)
for j, (m, c) in enumerate(zip(grp.index, [PAPER_C, "#93b4f5", OURS_C])):
    axes[1].bar(xx + (j - 1) * 0.27, grp.loc[m], 0.26, color=c, label=m)
axes[1].set_xticks(xx, ["users seen in training\n(16 % of test)", "unseen
users\n(84 % of test)"])
axes[1].set_ylim(0.65, 0.85); axes[1].set_ylabel("MAE");
axes[1].legend(frameon=False, fontsize=8)
axes[1].set_title("Where the gain comes from")
fig.tight_layout(); fig.savefig("results/fig_improvement.png",
bbox_inches="tight"); plt.show()
```





In [14]:
```
t5i = t5[["paper", "ours_mean", "improved_mean"]].rename(columns=
{"ours_mean": "paper model (ours)",

"improved_mean": "DMCCF+"})
display(Markdown("**Criteria-rating MAE (Table 5) with DMCCF+**"),
t5i.round(4))
```

Criteria-rating MAE (Table 5) with DMCCF+

	paper	paper model (ours)	DMCCF+
Value	0.8195	0.8400	0.8226
Rooms	0.7612	0.7879	0.7586
Location	0.6581	0.6875	0.5782
Cleanliness	0.6324	0.6644	0.6232
Check in / front desk	0.7582	0.7844	0.7495
Service	0.7401	0.7681	0.7337
Business service	0.8774	0.8996	0.8608
Mean	0.7495	0.7760	0.7324

What the improvement achieved

	Paper's model (reproduced)	Paper's reported number	DMCCF+
Overall MAE	0.777 ± 0.010	0.755	**0.737 ± 0.004**
Criteria MAE (mean of 7)	0.776	0.750	**0.732**
Predictions within 1 star	74.9 %	–	**75.4 %**
Exact star after rounding	47.9 %	–	**48.9 %**

- **5.0 % lower error than the paper's model.** It is better on all 5 folds (paired t-test p = 0.001), and also below the paper's own reported 0.755.
- **8.1 % lower error than SVD**, the best classical method (5/5 folds, p < 0.001).
- **Fold-to-fold variation halves** (std 0.004 against 0.010): no dead networks, no restarts, and a 5-seed average.

The gain comes from unseen users. For the 84 % of test ratings whose user is new, the error falls from 0.784 to **0.729**. The unknown-user token gives them a learned "typical reviewer" instead of a random vector.

The trade-off. For users seen in training, the error rises from 0.738 to 0.782. Swapping 30 % of IDs for the unknown token, plus a smaller embedding, means the

model relies less on a user's one or two previous reviews. Since these users are only 16 % of test ratings, the trade is clearly worth it overall. A natural next step is a separate path or gate for returning users.

What did not change. F1 and F2 stay at the paper model's level (0.865 / 0.860 against 0.866 / 0.858). These metrics only check which side of 4 stars a prediction falls on, and the two models agree there almost everywhere. MAP and MRR stay at about 0.998 for every method, for the reason given in finding 6.

8. Accuracy, not just error

MAE measures how far off a prediction is. The metrics below instead ask how often the model is right, computed on the same cached test predictions (`accuracy_metrics.py`):

- **Exact star / within 1 star:** the rounded prediction hits the true star / is at most 1 star off.
- **Good-hotel accuracy:** predicted ≥ 4 agrees with true ≥ 4. Because 77.5 % of ratings are ≥ 4, also shown as **balanced accuracy**, which counts good and not-good ratings equally.
- **ROC-AUC:** how well the predictions rank good stays above bad ones (0.5 = random).
- **R^2:** share of rating variance explained.

DMCCF+ blend is the plain 50/50 average of DMCCF+ and SVD. The weight was checked on the inner split, not tuned on the test folds.

In [15]:
```
import accuracy_metrics as A
A.main()
acc = pd.read_csv("results/accuracy_metrics.csv", index_col=0)
tests = pd.read_csv("results/accuracy_tests.csv")
display(Markdown("**5-fold test, mean** (MAE lower is better; everything
else higher is better)"), acc.round(4))
display(Markdown("**Against the paper's model, paired over 5 folds**"),
tests.round(4))
```

```
                          MAE  exact star  within 1 star  good-hotel acc
balanced acc  ROC-AUC      R2
method
SVD                    0.8025      0.4036         0.7347          0.6937
0.6720   0.7322  0.1790
Paper model            0.7765      0.4786         0.7491          0.7348
0.6407   0.7100  0.0942
DMCCF+                 0.7374      0.4892         0.7543          0.7308
0.6251   0.7077  0.0348
DMCCF+ blend (with SVD)  0.7657      0.4806         0.7397          0.7022
0.6662   0.7286  0.1494
always predicting 'good' scores 0.7747

                 method         metric   vs paper model  better folds
p
                 DMCCF+            MAE 0.7765 -> 0.7374             5
0.0010
                 DMCCF+     exact star 0.4786 -> 0.4892             5
0.0320
                 DMCCF+  within 1 star 0.7491 -> 0.7543             4
0.1740
                 DMCCF+ good-hotel acc 0.7348 -> 0.7308             2
0.5857
                 DMCCF+   balanced acc 0.6407 -> 0.6251             0
0.0371
                 DMCCF+        ROC-AUC 0.7100 -> 0.7077             1
0.1001
                 DMCCF+             R2 0.0942 -> 0.0348             0
0.0003
DMCCF+ blend (with SVD)            MAE 0.7765 -> 0.7657             4
0.0626
DMCCF+ blend (with SVD)     exact star 0.4786 -> 0.4806             3
0.5262
DMCCF+ blend (with SVD)  within 1 star 0.7491 -> 0.7397             0
0.0014
DMCCF+ blend (with SVD) good-hotel acc 0.7348 -> 0.7022             0
0.0011
DMCCF+ blend (with SVD)   balanced acc 0.6407 -> 0.6662             5
0.0006
DMCCF+ blend (with SVD)        ROC-AUC 0.7100 -> 0.7286             5
0.0000
DMCCF+ blend (with SVD)             R2 0.0942 -> 0.1494             5
0.0010
```

5-fold test, mean (MAE lower is better; everything else higher is better)

method	MAE	exact star	within 1 star	good-hotel acc	balanced acc	ROC-AUC	R2
SVD	0.8025	0.4036	0.7347	0.6937	0.6720	0.7322	0.1790
Paper model	0.7765	0.4786	0.7491	0.7348	0.6407	0.7100	0.0942
DMCCF+	0.7374	0.4892	0.7543	0.7308	0.6251	0.7077	0.0348
DMCCF+ blend (with SVD)	0.7657	0.4806	0.7397	0.7022	0.6662	0.7286	0.1494

Against the paper's model, paired over 5 folds

	method	metric	vs paper model	better folds	p
0	DMCCF+	MAE	0.7765 -> 0.7374	5	0.0010
1	DMCCF+	exact star	0.4786 -> 0.4892	5	0.0320
2	DMCCF+	within 1 star	0.7491 -> 0.7543	4	0.1740
3	DMCCF+	good-hotel acc	0.7348 -> 0.7308	2	0.5857
4	DMCCF+	balanced acc	0.6407 -> 0.6251	0	0.0371
5	DMCCF+	ROC-AUC	0.7100 -> 0.7077	1	0.1001
6	DMCCF+	R2	0.0942 -> 0.0348	0	0.0003
7	DMCCF+ blend (with SVD)	MAE	0.7765 -> 0.7657	4	0.0626
8	DMCCF+ blend (with SVD)	exact star	0.4786 -> 0.4806	3	0.5262
9	DMCCF+ blend (with SVD)	within 1 star	0.7491 -> 0.7397	0	0.0014
10	DMCCF+ blend (with SVD)	good-hotel acc	0.7348 -> 0.7022	0	0.0011
11	DMCCF+ blend (with SVD)	balanced acc	0.6407 -> 0.6662	5	0.0006
12	DMCCF+ blend (with SVD)	ROC-AUC	0.7100 -> 0.7286	5	0.0000
13	DMCCF+ blend (with SVD)	R2	0.0942 -> 0.1494	5	0.0010

What this shows. No single model wins every metric:

- **Error-type accuracy: DMCCF+ is best.** It has the lowest MAE, the most exact stars (48.9 % vs 47.9 %, better on 5/5 folds, p = 0.03) and the most predictions within 1 star.
- **Ranking-type accuracy: SVD and the blend are best.** On ROC-AUC, balanced accuracy and R^2, DMCCF+ is no better than the paper's model. Training on absolute error pulls predictions toward the typical 4–5 stars, which blurs which stays are worse. The blend fixes that: ROC-AUC 0.710 → **0.729**, balanced accuracy 64.1 % → **66.6 %**, R^2 0.094 → **0.149**, each better on 5/5 folds, while its MAE (0.766) is still below the paper model's.
- **Plain "good hotel?" accuracy is not a useful measure here.** Always answering "good" scores 77.5 %, more than any model, because good stays dominate.

So use DMCCF+ to predict the star rating, and the blend to rank hotels.