# Embedding and ranking for images, entities, items and text.


**Jason Weston**

**Facebook, NY**

(However, this work was done while I was at Google.)


Collaborators: Samy Bengio, Maya Gupta, Aurelien Lucchi, Antoine Bordes, Nicolas Usunier, Oksana Yakhnenko, Ron Weiss, Hector Yee and Ameesh Makadia.

# Large Scale Ranking, Retrieval and Recommendation

Tasks:

- Retrieve text documents given a query string (Web Search).
- Retrieve images given a query string (Image Search).
- Rank possible annotations given an image (Google Goggles or Glass).
- Recommend videos given a user profile or query video (YouTube).

Use *machine learning* over huge datasets to improve the tasks:

- Millions or billions of (noisily) labeled examples via user clicks.
- Millions or billions of items to rank (documents, images, videos).
- Millions of features to learn from (words, n-grams, user profiles e.g. videos watched).

# Embedding and Ranking Models

Supervised embedding and ranking models (e.g. Wsabie) can be used.
In this talk:

- Image Annotation.
- Image Search.
- YouTube or Google Music recommendation.

Issues we'd like to address:

- What's the right ranking loss function? How do we do use SGD?
- Enough nonlinearity to model the problem? How do we do use SGD?

*This talk will be in several sections that discuss these issues..*

# Image Annotation (& Image Search is similar)

Goal: Rank labels given an image: 100,000s+ of possible annotations.

barrack obama,
barak obama,
barack hussein obama,
barack obama,
james marsden,
jay z,
nelly

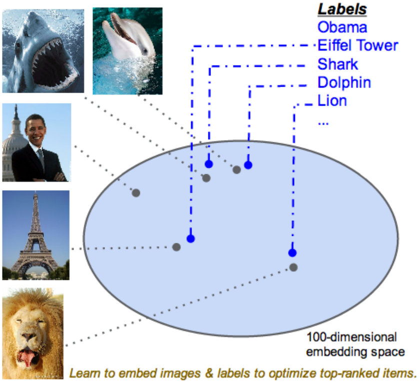
eiffel,
paris by night,
la tour eiffel,
tour eiffel,
eiffel tower,
las vegas strip,
tokyo tower

Datasets we use in some reported experiments:

| Statistics | ImageNet 16k* | Web Images |
|---|---|---|
| Number of Examples | 9 Million | 50 Million |
| Number of Labels | 15,589 | 96,812 |

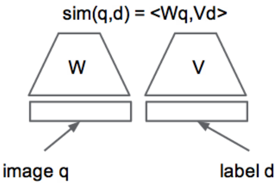
# Wsabie: Word Embeddings for Images [Weston et al., '10]

## First Application: Image Annotation




- Train the model on Image Search click data.
- We use stochastic gradient descent
- Single machine (multi-core)
- We use 50M training examples, 10k image features, and 100k labels (queries).
- We want:

sim( , dolphin) > sim( , obama) + 1



# Wsabie: AUC Loss Training

## Training Loss

- Ranking loss from preference triplets $(x, y^+, y^-)$,
  *"for query $x$, result $y^+$ should appear above $y^-$"*:

**Classical approach to learning to rank is maximize AUC by minimizing:**

$$\sum_x \sum_y \sum_{\bar{y} \neq y} |1 + f_{\bar{y}}(x) - f_y(x)|_+$$

**Learning Algorithm** Stochastic Gradient Descent:

Iterate | Sample a triplet $(x, y^+, y^-)$,
Update $W \leftarrow W - \lambda \frac{\partial}{\partial W} \max(0, 1 - f_{y^+}(x) + f_{y^-}(x))$.

Other things we use: adagrad, parallel SGD (hogwild), ...

# Ranking Annotations: AUC is Suboptimal

**Classical approach to learning to rank is maximize AUC by minimizing:**

$$\sum_{x} \sum_{y} \sum_{\bar{y} \neq y} |1 + f_{\bar{y}}(x) - f_y(x)|_+$$

Problem: All pairwise errors are considered the same, it counts the number of ranking violations.

Example:

Function 1: true annotations ranked 1st and 101st.

Function 2: true annotations ranked 50th and 52nd.

AUC prefers these *equally* as both have 100 "violations".

We want to optimize the top of the ranked list!

# Ordered Weighted Pairwise Classification (OWPC) Loss

**A class of ranking error functions defined in [Usunier et al. '09]:**

$$err(f(x), y) = L(rank_y(f(x))),$$

where

$$L(k) = \sum_{j=1}^{k} \alpha_j, \text{ with } \alpha_1 \geq \alpha_2 \geq \cdots \geq 0.$$

Here $rank_y(f(x))$ is the rank of the true label $y$ given by $f(x)$:

$$rank_y(f(x)) = \sum_{\bar{y} \neq y} I(f_{\bar{y}}(x) \geq f_y(x))$$

Different choices of $L(\cdot)$ have different minimizers:

$\alpha_j = \frac{1}{Y-1} \rightarrow$ minimize mean rank

$\alpha_j = \frac{1}{j} \rightarrow$ more weight on optimizing the top of list.

Example from before: $\alpha_j = \frac{1}{j} \rightarrow$ err(func1)=5.18, err(func2)=8.99.

# Weighted Approximate-Rank Pairwise (WARP) Loss

**Problem: we would like to apply SGD:**

$$\textit{Weighting } L(rank_y(f(x))), \qquad rank_y(f(x)) = \sum_{\bar{y} \neq y} I(f_{\bar{y}}(x) + 1 \geq f_y(x))$$

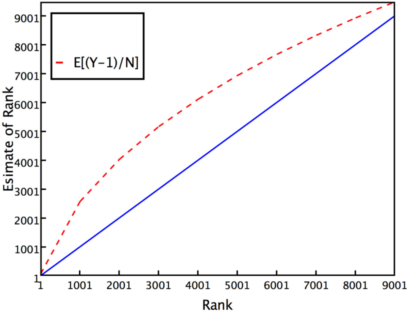
. . . too expensive to compute per $(x, y)$ sample as $y \in \mathcal{Y}$ is large.

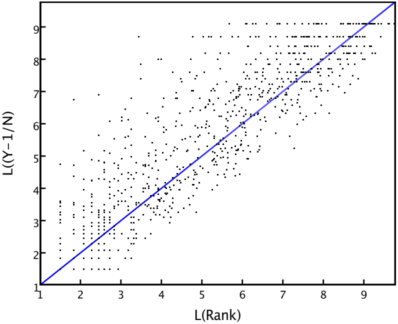
**Solution: approximate by sampling $f_i(x)$ until we find a violating label $\bar{y}$**

$$rank_y(f(x)) \approx \left\lfloor \frac{|\mathcal{Y}| - 1}{N} \right\rfloor$$

where $N$ is the number of trials in the sampling step.

# WARP Loss : Approximation Accuracy




$$E[N] = \sum_{i=1}^{\infty} i(1-p)^{i-1}p,$$

$$E[\frac{(Y-1)}{N}] = \sum_{i=1}^{\infty} \frac{Y-1}{i}(1-p)^{i-1}p$$

$$p = Pr(violation) = \frac{rank}{Y-1}$$

On average, our approximation tends to overestimate.
However it corresponds to another choice of $L$, so adjust $\alpha$ accordingly.

# Online WARP Loss

**Input:** labeled data $(x_i, y_i)$, $y_i \in \{1, \ldots, Y\}$.

**repeat**

  Pick a random labeled example $(x_i, y_i)$

  (*assume only one true label, else also select randomly from positive labels*)

  Set $N = 0$.

  **repeat**

    Pick a random annotation $\bar{y} \in \{1, \ldots, Y\} \setminus y_i$.

    $N = N + 1$.

  **until** $f_{\bar{y}}(x) > f_{y_i}(x) - 1$ or $N > Y - 1$

  **if** $f_{\bar{y}}(x) > f_{y_i}(x) - 1$ **then**

    Make a gradient step to minimize:

$$L(\lfloor \frac{Y-1}{N} \rfloor)|1 - f_y(x) + f_{\bar{y}}(x)|_+$$

    Constrain $||U_i|| \leq 1$, $||V_i|| \leq 1$.

  **end if**

**until** validation error does not improve.

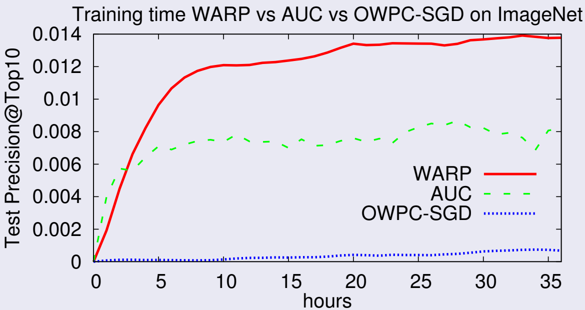
# Related Work

- SVM [Joachims, 2002] and NN ranking methods [Burges, 2005] .
  Use hand-coded features: title, body, URL, search rankings,. . .
  (e.g. Burges uses 569 features in all).
- In contrast we can use millions of features, e.g. words or videos, and try to find their hidden representation.
- Many works on optimizing different loss functions (MAP, ROC, NDCG): [Cao, 2008], [Yu, 2007], [Qin, 2006],. . . .
- [Grangier & Bengio, '06] supervised methods to for retrieving images but are full rank (cannot handle as many features).
- [Goel, Langord & Strehl, '08] used Hash Kernels (Vowpal Wabbit).
- Unsupervised methods like LSI also handle millions of features, but do not work as well. SVD usually not as good either . . .
- See also earlier NEC work with colleagues there e.g. [Bai et al. '09].

## Image Annotation Performance

| Algorithm | 16k ImageNet | 22k ImageNet | 97k Web Data |
|---|---|---|---|
| Nearest Means | 4.4% | 2.7% | 2.3% |
| One-vs-all SVMs 1+:1- | 4.1% | 3.5% | 1.6% |
| One-vs-all SVMs | 9.4% | 8.2% | 6.8% |
| AUC Embedding | 4.7% | 5.1% | 3.1% |
| Wsabie (WARP Embedding) | 11.9% | 10.5% | 8.3% |

## Training time: WARP vs. OWPC-SGD & AUC



# Learned Annotation Embedding (on Web Data)

| Annotation | Neighboring Annotations |
|---|---|
| barack obama | *barak obama*, obama, barack, barrack obama, bow wow |
| david beckham | *beckham*, *david beckam*, alessandro del piero, *del piero* |
| santa | *santa claus*, *papa noel*, *pere noel*, santa clause, joyeux noel |
| dolphin | delphin, dauphin, *whale*, *delfin*, *delfini*, *baleine*, blue whale |
| cows | *cattle*, *shire*, dairy cows, kuh, *horse*, cow, *shire horse*, *kone* |
| rose | rosen, *hibiscus*, *rose flower*, rosa, roze, pink rose, *red rose* |
| pine tree | *abies alba*, abies, *araucaria*, pine, neem tree, *oak tree* |
| mount fuji | mt fuji, fuji, fujisan, fujiyama, *mountain*, *zugspitze* |
| eiffel tower | *eiffel*, tour eiffel, la tour eiffel, *big ben*, *paris*, blue mosque |
| ipod | i pod, *ipod nano*, apple ipod, ipod apple, new ipod |
| f18 | f 18, eurofighter, f14, fighter jet, tomcat, mig 21, f 16 |

# Wikipedia Experiments: Document Retrieval Performance

[Bai et al. '09] did some "toy" experiments on Wikipedia, which still contains 2 million documents. We set up a retrieval task using the link structure and separated the data into 70% for training and 30% for test.

Document based retrieval:

| Algorithm | Rank-Loss | MAP | P10 |
|---|---|---|---|
| TFIDF | 0.842% | 0.432±0.012 | 0.193 |
| $\alpha$LSI + $(1-\alpha)$TFIDF | 0.721% | 0.433 | 0.193 |
| Linear SVM Ranker | 0.410% | 0.477 | 0.212 |
| Hash Kernels + $\alpha I$ | 0.322% | 0.492 | 0.215 |
| Wsabie + $\alpha I$ (AUC) | 0.158% | 0.547±0.012 | 0.239±0.008 |

*k*-keywords based retrieval:

| $k=5$: Algorithm | Params | Rank | MAP | P@10 |
|---|---|---|---|---|
| TFIDF | 0 | 21.6% | 0.047 | 0.023 |
| $\alpha$LSI + $(1-\alpha)$TFIDF | $200\mathcal{D}+1$ | 14.2% | 0.049 | 0.023 |
| Wsabie + $\alpha I$ (AUC) | $400\mathcal{D}$ | **4.37%** | **0.166** | **0.083** |

# Scatter Plots: Wsabie vs. TFIDF and LSI

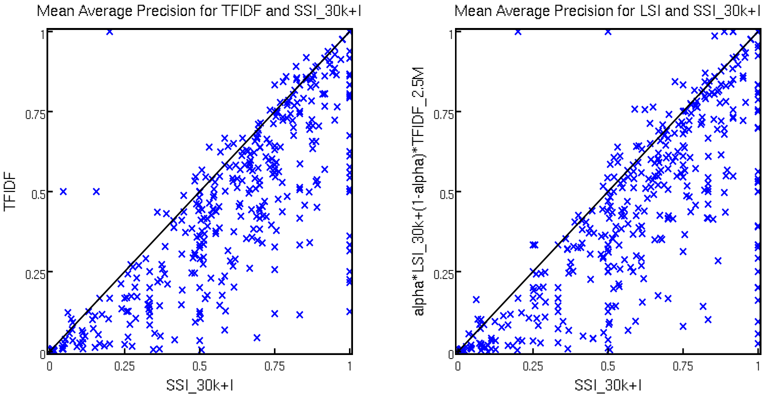



Figure : Scatter plots of Average Precision for 500 documents:
(a) SSI vs. TFIDF, (b) SSI vs. $\alpha$LSI + $(1 - \alpha)$ TFIDF.

# Learning to Rank Recommendations with the $K$-Order Statistic Loss

**Jason Weston, Hector Yee, Ron Weiss**

**Google, USA**

# Summary

In this work we consider **recommendation** as a **ranking** problem.
Explore the question: *what is the right ranking loss function to use?*

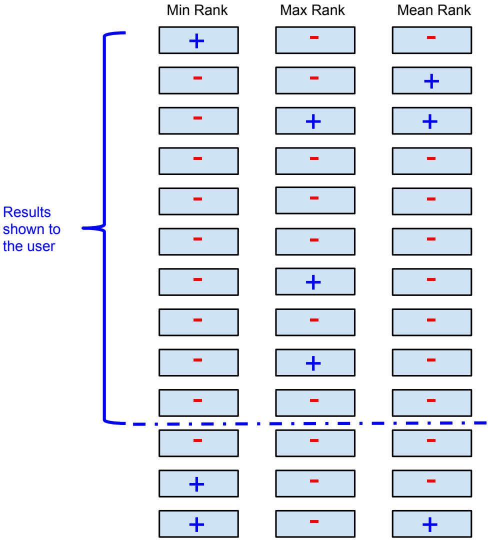
We develop a new family of losses: $K$-Order Statistic Loss.

It includes as special cases some existing ranking losses (AUC, WARP).
It also suggests new variants:

- Optimize p@k better than WARP.
- Optimize mean max rank – *good for "diversity"*
- Variants in-between

We give results on Google Music and YouTube recommendations.

K-OS loss is flexible: change sampling strategy of +ves and -ves in SGD.




Diversity (Max Rank) type approach gave $\approx$+1% watch time for YouTube.

# Training matrix factorization models by SGD

## Matrix Factorization Models

Consider general MF model: $f_y(x) = x^\top (U^\top V) y$,

- $x \in \mathbb{R}^d$ are features of the user, $y \in \mathbb{R}^d$ are features of a given item.

Special case 1: standard MF model $U_u^\top V_d$ for user $u$ and item $d$

Special case 2: user represented as set of items 'liked': $\frac{1}{|\mathcal{Y}_x|} \sum_{i \in \mathcal{Y}_x} V_i^\top V_d$,

Train by stochastic gradient descent (SGD):

- Repeat
- Pick a random user.
- Do an SGD step for this user.
- Until validation error is lowest.

..where SGD step depends on the loss function... *(next slide!)*

# New! $K$-OS ($K$-Order Statistic) Loss

Existing Loss functions:

- AUC: optimize mean rank
- WARP: approximately optimizes top of list (precision@k)

$K$-OS loss generalizes old approaches and also can do new things:

- can more accurately optimize precision at $k$.
- optimize things like mean maximum rank.

*Motivation: user watches 80% classical music videos, and 20% about polar bears. We can optimize p@1 by putting the classical at the top, and giving up on the bears. Or we can care where the lowest bear is.*

**How? During SGD step for a user:**

- Sample $K$ positive items from user.
- Order them by the score assigned by the model.
- Weight the SGD update for positives as a function of this ordered set.

## $K$-OS loss

The $k$-OS loss can be written as:

$$L_{K\text{-OS}}(f(x), \mathcal{Y}_x)) = \frac{1}{Z} \sum_{i=1}^{|\mathcal{Y}_x|} P\Big(\frac{i}{|\mathcal{Y}_x|}\Big) \Phi\Big(rank_{\mathcal{Y}_{x_{o_i}}}(f(x))\Big)$$

where $Z = \sum_i P\Big(\frac{i}{|\mathcal{Y}_x|}\Big)$ normalizes.

$P(\frac{j}{100})$ is the weight for the $j^{th}$ percentile of the ordered positive items.

## Different choices of $P$ result in different loss functions:

- $P(j) = C$ for all $j$ gives the WARP or AUC loss.
- $P(i) > P(j)$ for $i < j$ result in paying more attention to positive items that are at the top of the ranked list, ignores lower ranked positives.
- $P(i) < P(j)$ for $i < j$ focuses more on improving the worst ranked positives in the user's rating set. We hypothesize this better captures all the user's tastes – measure this using the mean max rank metric.

# $K$-OS Loss

**repeat**

Pick a user $x$ at random from the training set.

Pick $i = 1, \ldots, K$ positive items $d_i \in \mathcal{Y}_x$.

Compute $f_{d_i}(x)$ for each $i$.

Sort the scores by descending order, let $o(j)$ be the index into $d$ that is in position $j$ in the list

$$f_{d_{o_1}}(x) > f_{d_{o_2}}(x) > \cdots > f_{d_{o_K}}(x).$$

Pick a position $k \in 1, \ldots, K$ using some known distribution $P$.

Perform AUC or WARP step with that positive $d_{o_k}$.

**until** validation error does not improve.

Simplest distribution $P$: always pick the same fixed position $k$.

*E.g. if always pick the first position ($k = 1$) optimizes p@1.*

*Or, if always pick the last position ($k = K$) optimizes mean max rank.*

# K-OS loss: results

| Dataset | Music: Artists | Music: Tracks | YouTube |
|---|---|---|---|
| Number of Items | ≈75k | ≈700k | ≈500k |
| Train Users | Millions | | |
| Test Users | Tens of Thousands | | |

## Google Music Artist Recommendation

| Method | Mean Rank | Max Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|---|
| SVD | +187% | +205% | -19% | -14% | -19% | -14% |
| WARP | - | - | - | - | - | - |
| K-OS k=1 | +195% | +224% | -1.3% | -5.2% | -1.8% | -5.6% |
| K-OS k=2 | +88% | +110% | +1% | -0.4% | +0.7% | -0.6% |
| K-OS k=3 | +23% | +32% | -1% | -0.4% | -1.6% | -0.4% |
| K-OS k=4 | -7.7% | -6.4% | -3% | -2% | -4% | -2% |
| K-OS k=5 | -16% | -18% | -14% | -7% | -14% | -7% |

# K-os loss: results

## Google Music Artist Recommendation - AUC baseline

| Method | Mean Rank | Max Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|---|
| SVD | +254% | +284% | +1.6% | -2.5% | +0.72% | -2.1% |
| WARP | +23% | +26% | +25% | +14% | +25% | +13% |
| AUC | - | - | - | - | - | - |
| K-os k=1 | +159% | +194% | +1.3% | -5% | -0.1% | -5.6% |
| K-os k=2 | +65% | +80% | +9% | +0.3% | +7% | -0.4% |
| K-os k=3 | +15% | +20% | +10% | +3.9% | +9% | +3.6% |
| K-os k=4 | -2.7% | -1.6% | +6% | +3% | +5.9% | +2.7% |
| K-os k=5 | -2.2% | -3.7% | -25% | -8% | -24% | -8% |

## Google Music Track Recommendation

| Method | Mean Rank | Max Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|---|
| WARP | - | - | - | - | - | - |
| K-OS k=1 | +323% | +271% | +16% | +3.3% | +17% | +4.3% |
| K-OS k=2 | +209% | +199% | +22% | +14% | +23% | +15% |
| K-OS k=3 | +50.7% | +61% | +22% | +19% | +22% | +20% |
| K-OS k=4 | -44.1% | -40.9% | +9.1% | +15% | +12% | +16% |
| K-OS k=5 | -54.7% | -54.8% | -50% | -32% | -50% | -33% |

## YouTube Video Recommendation (also ∼1% clickthrough gain in live exp.)

| Method | Mean Rank | Max Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|---|
| SVD | +56% | +45.3% | -54% | -57% | -54% | -96% |
| WARP | - | - | - | - | - | - |
| K-OS k=1 | +119% | +101% | +14% | +6.5% | +12% | +3.5% |
| K-OS k=2 | +55% | +71% | +7% | +6% | +5.8% | +4.8% |
| K-OS k=3 | +10% | +19% | -1.4% | +1.3% | -1.2% | +2.1% |
| K-OS k=4 | -10% | -13% | -10% | -6.4% | -8% | -3.8% |
| K-OS k=5 | -14% | -23% | -36% | -32% | -34% | -30% |

# Conclusion

- New class of loss functions: $K-$OS ($K$-Order Statistic) Loss.
- Highly scalable.
- Flexible to optimize different metrics, depending on what you want.
- "Diversity" metric (optimizing smoothed "mean max rank") worked well in live experiments on YouTube.

Future work: *could generalize new loss: sample over users* **and** *items?*

# Nonlinear Latent Factorization by Embedding Multiple User Interests

**Jason Weston, Ron Weiss, Hector Yee**

**Google, USA**

# Motivation: problem

YouTube recommendations: a user is interested in videos of classical music and nature videos and ... – we might want to model that.


Standard embedding models of the form:

$$f(u, d) = U_u^\top V_d$$

User $u$ is modeled with e.g. 100 dim vector $U_u$.
Item $d$ is also modeled with e.g. 100 dim vector $V_d$.

**Hypothesis:**

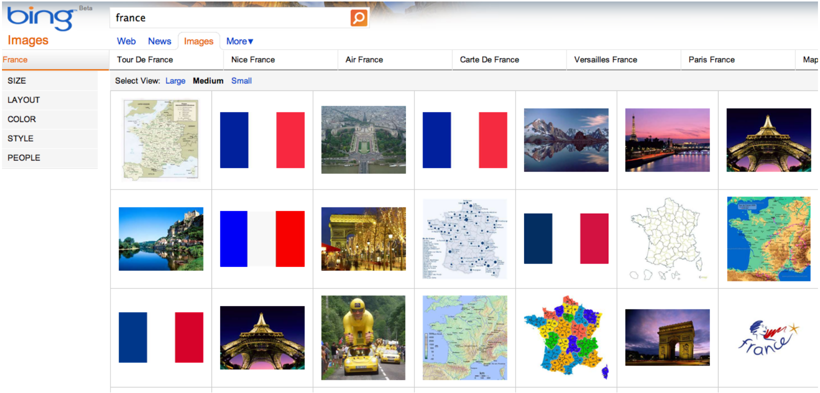
User is a more complicated entity than any given item.
Can't model all user's item tastes in same dimensionality as items.

**In this work, we design a factor model for multiple user tastes.**

# Motivation: algorithm

[Lucchi & Weston, ECCV'12] developed a model for image search to deal with ambiguous queries.

- Example: the query France has relevant images from distinct senses: *the flag, maps, images of cities (such as Paris) and monuments . . .*



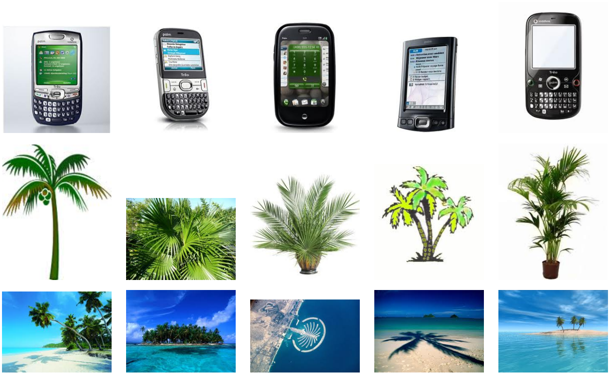

- Model: $f(query, image) = \max_{sense \in Senses(query)} W_{query,sense} \cdot \Phi(image)$

# Learnt senses of the query “palm”

$s = 1$

$s = 2$

$s = 3$



Senses learnt for query “palm”:

| | |
|---|---|
| **palm** $s = 1$ | blackberry, lg gd900, future phones, blackberry 9800, blackberry 9800 torch, smartphone, blackberry curve, nokia e, nokia phones, lg phones, cellulari nokia, nokia, nokia mobile phones, blackberry pearl, nokia mobile, lg crystal, smartphones. |
| **palm** $s = 2$ | palmier, palm tree, coconut tree, money tree, dracaena, palme, baum, olive tree, tree clip art, tree clipart, baobab, dracena, palma, palm tree clip art, palmera, palms, green flowers, palm trees, palmeras. |
| **palm** $s = 3$ | palmenstrand, beautiful beaches, playas paradisiacas, palms, beaches, lagoon, tropical beach, maldiverna, polinesia, tropical beaches, beach wallpaper, beautiful beach, praias, florida keys, paisajes de playas, playas del caribe, ocean wallpaper, karibik, tropical islands, playas. |

# MaxMF: applying a similar model for recommendation

Collaborative filtering: standard user-item factorizations are of the form:

$$f(u, d) = U_u^\top V_d$$

We generalize that to the following model:

$$f(u, d) = \max_{i=1,\dots,T} \hat{U}_{iu}^\top V_d.$$

where we model each user with $T$ tastes.
(Note $T = 1$ gives us the standard model.)

*Related nonlinear/taste modeling works:*
[Lawrence et al.,'09], [Salakkhutdinov et al.,'07], [Baltrunas et al.,'09],. . .
*(see paper for more details).*

# MaxMF: Large Scale MapReduce training

For 100s of millions of users SGD might have problems (e.g. won't fit in memory). One could consider a simpler (lower memory) model:

$$f_1(u, d) = \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} V_i^\top V_d$$

(Instead of $f(u, d) = U_u^\top V_d$.), where $\mathcal{D}_u$ is the set of positive items for user $u$, *i.e. we model each user as a linear combination of item embeddings.*

**To train our model one can then do the following:**

Train the model $f_1(u, d) = \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} V_i^\top V_d$ above with SGD.
Define $f_2(u, d) = \max_i \hat{U}_{iu}^\top V_d^*$, where $V^* = V$ from $f_1$.
**for** each user $u$ (in parallel) **do**
    Train $\hat{U}_{iu}$, but keep $V^*$ fixed.
    These can be done independently $\rightarrow$ use MapReduce.
**end for**

# MaxMF: experiments

## Datasets

| Dataset | Music: Artists | Music: Tracks | YouTube |
|---|---|---|---|
| Number of Items | ≈75k | ≈700k | ≈500k |
| Train Users | Millions | | |
| Test Users | Tens of Thousands | | |

- For each test user, we leave out 5 random items from training, and see where they are ranked at test time (against all unrated items).
- We measure mean rank, prec@n and rec@n. We report relative changes to MaxMF $T = 1$ baseline (i.e, standard ranker).
- We also compare to SVD: *L2-optimal matrix factorization for the complete matrix with log-odds weighting on the columns, which downweights the importance of the popular features, as that worked better than uniform weights.*

# MaxMF: Google Music Results

## Google Music Artist Recommendation

| Method | Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|
| SVD | +26% | -7.9% | -1.5% | -6.7% | -0.75% |
| MaxMF T=1 | - | - | - | - | - |
| MaxMF T=3 | -3.9% | -3.1% | -0.18% | -4.4% | -0.15% |
| MaxMF T=5 | -8% | -0.46% | +1.3% | -0.63% | +1.9% |
| MaxMF T=10 | -11% | +0.33% | +3.1% | +0.33% | +3.2% |

## Google Music Track Recommendation

| Method | Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|
| MaxMF T=1 | - | - | - | - | - |
| MaxMF T=2 | -7.7% | +10% | +12% | +9.8% | +10% |
| MaxMF T=3 | -12% | +20% | +21% | +20% | +20% |
| MaxMF T=10 | -17% | +30% | +32% | +33% | +34% |

# MaxMF: YouTube results

## YouTube Video Recommendation

| Method | Rank | P@1 | P@10 | R@1 | R@10 |
|---|---|---|---|---|---|
| SVD | +56% | -54% | -57% | -54% | -96% |
| MaxMF T=1 | - | - | - | - | - |
| MaxMF T=2 | -3.2% | +8.8% | +13% | +9.9% | +14% |
| MaxMF T=3 | -6.2% | +16% | +18% | +17% | +19% |
| MaxMF T=5 | -9% | +22% | +23% | +23% | +23% |
| MaxMF T=10 | -11% | +26% | +26% | +28% | +26% |

# Conclusion

- New nonlinear embedding approach to model $T$ user tastes.
- We used it with a ranking loss (WARP), but other losses are possible.
- Scalable training for 100M+ users via MapReduce.
- Improved results on YouTube & Google Music.

Future work:

- Choose $T$ per user, e.g. based on (number of) positives?
- Smooth the max by weighting w.r.t the order instead – might be more robust.

# Summary

Supervised embedding models can be useful in a bunch of tasks:

- Image Annotation.
- Image Search.
- YouTube or Google Music recommendation.

Issues we tried to address:

- What's the right ranking loss function? ($K$-os loss)
- Enough nonlinearity to model the problem? (MaxMF)