# Affinity Weighted Embedding

**Jason Weston** JWESTON@GOOGLE.COM
**Ron Weiss** RONW@GOOGLE.COM
Google Inc, New York, NY, USA

**Hector Yee** HYEE@GOOGLE.COM
Google Inc, San Bruno, CA, USA.

## Abstract

Supervised linear embedding models like WSABIE (Weston et al., 2011) and supervised semantic indexing (Bai et al., 2010) have proven successful at ranking, recommendation and annotation tasks. However, despite being scalable to large datasets they do not take full advantage of the extra data due to their linear nature, and we believe they typically underfit. We propose a new class of models which aim to provide improved performance while retaining many of the benefits of the existing class of embedding models. Our approach works by reweighting each component of the embedding of features and labels with a potentially nonlinear affinity function. We describe several variants of the family, and show its usefulness on several datasets.

## 1. Introduction

Given an initial representation of data, one way to perform learning is to find the parameters of a function that maps that data to another, possibly lower-dimensional space – the so-called *embedding space*. By measuring similarity in the learnt embedding space, one can solve a variety of tasks, such as classification, retrieval, ranking, recommendation or regression. Linear embedding models have had a high degree of success in all these applications, for example singular value decomposition (SVD) for recommendation (Billsus & Pazzani, 1998; Koren et al., 2009), latent semantic indexing (LSI) for retrieval (Deerwester et al., 1990) and newer methods like supervised semantic indexing (Bai et al., 2010) and WSABIE which use a supervised ranking objective instead (Weston et al., 2011; Bai et al., 2010). These models are simple to understand and implement, and are scalable to large training sets in terms of number of examples and features due to the low dimensionality of the embedding. However, despite their success, their performance largely depends on the quality of the original features and due to their linear nature they can tend to underfit (Bai et al., 2010).

Our goal in this paper is to develop a class of embedding models that retains the beneficial qualities of standard linear embedding models while trying to improve on their shortcomings. We desire models that have high enough capacity to capture things of interest, are trained easily (have reliable optimization that scales to large data), are fast enough at test time, and if possible are simple to understand and implement as well.

The proposed class of models are called affinity weighted embedding (AWE) models. Analogous to, but not the same as, how kernel methods generalize linear models, we allow the user to define an affinity function. Model inference works by reweighting, using the affinity function, each part of the sum that comprises the embedding of input features and labels, thus giving a potentially nonlinear model. Similarly to kernel methods, one is free to define/engineer the affinity function that works best for a specific task or dataset. This gives both modeling power and control to the designer. We also explore the possibility of learning affinity functions rather than fixing them in advance. Suitable choices of affinity function can lead to higher capacity models that outperform linear models, and sometimes to faster models at inference time as well. We give results on a variety of datasets showing the usefulness of these various choices – for music annotation, image annotation and YouTube video recommendation.

The rest of the paper is organized as follows. In section 2 we give an overview of classical linear embedding models, and in section 3 we describe our proposed approach which is a generalization of them. We then give examples of affinity functions and describe training methods. In section 4 we describe related work and in section 5 we detail the experi-

*Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

ments we performed. Finally, section 6 concludes.

## 2. Linear Embedding Models

Standard linear embedding models are of the form:

$$f_{EMB}(x, y) = x^\top U^\top V y = \sum_{ij} x_i U_i^\top V_j y_j. \quad (1)$$

where $x \in \mathbb{R}^{|\mathcal{X}|}$ are the input features and $y \in \mathbb{R}^{|\mathcal{Y}|}$ are the output features. The output is either a possible label (in the annotation case), a document (in the information retrieval case) or an item (in the recommendation case). In the annotation case the output is typically a "one-hot" vector of all zeros and a single one in the dimension that indicates that particular class, where there are $|\mathcal{Y}|$ classes. In the recommendation case the one-hot representation is also often used, and $|\mathcal{Y}|$ is the number of items. In the information retrieval case the document is often represented as a bag of words, so $\mathcal{Y}$ is the dictionary of words, and a particular document is then represented as a sparse vector, e.g. with tf-idf weighting.

To score a given input-output pair, both the input and the output are mapped into a $d$-dimensional embedding space using the $d \times |\mathcal{X}|$ and $d \times |\mathcal{Y}|$ matrices $U$ and $V$ and then similarity in the embedding space is performed using a dot product. In some setups, other metrics such as Euclidean distance or cosine similarity are also used. At prediction time, one computes the above score $f(x, y)$ for each possible label, and then sorts (or otherwise keeps the top $k$ labels, depending on the task), largest score first.

These types of models are used in a variety of applications. For example, in the task of collaborative filtering, one is required to rank items according to their similarity to the user, and methods which learn latent representations of both users and items have proven very effective. In particular, singular value decomposition (SVD) (Billsus & Pazzani, 1998; Koren et al., 2009) and non-negative matrix factorization (NMF) (Lee & Seung, 2001) are two standard methods that at inference/prediction time use equation (1), although the methods to learn the actual parameters $U$ and $V$ themselves are different. In the task of document retrieval, on the other hand, one is required to rank text documents given a text query. The classical method latent semantic indexing (LSI) (Deerwester et al., 1990) is an unsupervised approach that learns from documents only, but still has the form of equation (1) at test time. Other popular methods include probabilistic latent semantic analysis (PLSA) (Hofmann, 1999) and latent dirichlet allocation (LDA) (Blei et al., 2003).

More recently, supervised methods have been proposed that learn the latent representation from (query, document) relevance pairs, e.g. the method supervised semantic indexing (SSI) (Bai et al., 2010). Finally, for multiclass classification tasks, particularly when involving thousands of possible labels, latent models have also proven to be very useful, e.g. the WSABIE model performs well on large-scale image (Weston et al., 2011) and music (Weston et al., 2012) annotation tasks. The latter algorithm also performs well for recommendation (Makadia et al., 2013; Weston et al., 2013).

Linear embedding models scale well to large data and are simple to implement and use. However, as they contain no nonlinearities, other than in the feature representations $x$ and $y$, they can be limited in their ability to fit large complex datasets, and typically seem to underfit, see e.g. (Bai et al., 2010) section 4.5 and figure 2.

## 3. Affinity Weighted Embedding Models

In this work we propose the following generalized embedding model:

$$f_{AWE}(x, y) = \sum_{ij} G_{ij}(x, y)\, x_i U_i^\top V_j y_j. \quad (2)$$

where $G$ is a function that measures the affinity between two points. Given a pair $x$, $y$ and feature indices $i$ and $j$, $G$ returns a scalar. Large values of the scalar indicate a high degree of match between feature $i$ of input $x$ and feature $j$ of label $y$.

If $G$ is chosen beforehand it can be seen as a way of encoding prior knowledge about these pairwise relationships analogous to choosing the kernel in kernel methods such as in support vector machines (SVMs) (Vapnik, 1998). However, in contrast to kernels, $G$ is not defined in the model as being computed over all training examples, but over all labels. If a particular feature on the input side, for a given label, is irrelevant then a good choice of $G$ would likely downweight it. Further, if a particular label (or feature of a label) is irrelevant for a given input, a good choice of $G$ would likely downweight that too. Similarly, relevant features or labels should be upweighted.

Potentially, $G$ can also be learnt rather than being specified beforehand, which can also be seen as related to the research area of learning kernels, see e.g. (Chapelle et al., 2002; Cortes et al., 2009).

Different methods of choosing (or learning) $G$ lead to different variants of our proposed approach. In the following subsection we will give some examples of specific choices one can employ. First, however, we describe some general recipes for constructing a function $G$:

- $G_{ij}(x, y) = c$ for some constant $c$. This is the baseline case. If $G$ returns a constant our model clearly reverts to the classical linear embedding model predictions using (1).

- $G_{ij}(x,y) = G(x,y)$. $G$ can be chosen such that it gives the same outputs for all $i$ and $j$ for a given pair $x, y$, i.e. $G_{ij}(x,y) = G(x,y)$. In this case each feature index pair $i, j$ returns the same scalar so the model reduces to:

$$f(x,y) = G(x,y)\, x^\top U^\top V y.$$

  In the special case of $\mathcal{X} = \mathcal{Y}$ one could use a known kernel function such as a radial basis function (RBF) (Shawe-Taylor & Cristianini, 2004) for example. For the more general case, when the dimensionality of the input and label are not the same, and other approaches must be employed.

- Another way to compare $x$ and $y$ is by running over a set of $m$ known training pairs and scoring based on those pairs:

$$G(x,y) = \sum_{i=1}^{m} K_x(x, \mathbf{x}_i) K_y(y, \mathbf{y}_i) \tag{3}$$

  where $\mathbf{x}$ and $\mathbf{y}$ are the sets of vectors from the training set, and $K_x$ and $K_y$ are kernel functions that operate on inputs and labels respectively. See the next subsection for concrete examples.

- $G_{ij}(x,y) = \mathbf{G}_{ij}$ where $\mathbf{G}$ is a $|\mathcal{X}| \times |\mathcal{Y}|$ matrix. In this case the returned scalar for $i$, $j$ is the same independent of the input vector $x$ and label $y$, i.e. it is a reweighting of the feature pairs. This gives the model:

$$f(x,y) = \sum_{ij} \mathbf{G}_{ij} x_i U_i^\top V_j y_j.$$

  This is likely only to be useful in large, possibly sparse, feature spaces, e.g. if $G_{ij}$ represents the weight of a word pair in an information retrieval task or an item pair in a recommendation task.

- Further, it is possible that $\mathbf{G}_{ij}$ could take a particular form, e.g. it is represented as a low rank matrix $\mathbf{G}_{ij} = \hat{U}_i^\top \hat{V}_j$. Or, for a simpler model, we set $\hat{U} = U$ and $\hat{V} = V$, i.e. to be the same as the parameters used in eq. (2). Finally, we can introduce a nonlinearity via an activation function $s : \mathbb{R} \mapsto \mathbb{R}$. Then, we have the function:

$$G_{ij}(x,y) = s(x_i U_i^\top V_j y_j). \tag{4}$$

  We describe some specific examples of this kind of function in the next subsection.

## 3.1. Specific Choices of $G_{ij}(x,y)$

We now describe some specific choices of $G$. In order for $G$ to produce a real-valued weight for its given arguments it is necessary for it to somehow compare $x$ and $y$ even though they may be of differing dimensionality.

In practice, we will explore two basic approaches to surmount this difficulty: (i) first map $x$ and $y$ to a common space and then perform a nonlinearity (cf. eq (4)); or (ii) compare $x$ and $y$ to known examples of the pairs independently, thus avoiding comparing $x$ and $y$ directly (cf. eq. (3)). We give specific examples of several possible choices of $G$ that follow one of the above two approaches.

**Sigmoid**

$$G_{i,j}(x,y) = s'(x_i U_i^\top V_j y_j) \tag{5}$$

where $s'$ is a sigmoid function, for example the logistic function:

$$s'(\eta) = \frac{1}{1+e^{-\eta}}.$$

This has the effect of switching off low scoring features, and only "activating" high scoring ones. It depends upon the matching score $x_i U_i^\top V_j y_j$ where $U$ and $V$ must be learnt appropriately (i.e. to activate and to switch off the right features for good performance).

**Linear Sigmoid** A similar choice is the linear sigmoid which approximates a sigmoid with a piece-wise linear function:

$$G_{i,j}(x,y) = s_l(x^\top U^\top V y) \tag{6}$$

where

$$s_l(\eta) = \begin{cases} \beta & \text{if } \eta < \beta, \\ \eta & \text{if } \beta \le \eta \le \alpha, \\ \alpha & \text{otherwise.} \end{cases}$$

Here, the parameters $\alpha$ and $\beta$ must also be chosen.

**Max** The max affinity is defined as:

$$G_{max}(x,y) = \max_i x_i U_i^\top V y. \tag{7}$$

This has the effect of taking the input feature with the maximum affinity with the output label being considered and reweighting the overall score using that weight. In a recommendation task, it thus could promote labels with a close nearest neighbor in the input. Consider for example a task like music recommendation, if the user has listened to a song very similar to the target output, a large weight will be used.

**Top-$k$** The max affinity can be generalized to the top-$k$ affinity, whereby the weight is assigned as the sum of the $k$ most similiar input features to give a smoother weighting function.

The affinity function is given by:

$$G_{top-k}(x, y) = \sum_{j \leq k} x_{t_j} U_{t_j}^\top V y.$$

where $t$ indexes the scores of the inputs such that they are sorted:

$$(x_{t_1} U_{t_1}^\top V y) \geq (x_{t_2} U_{t_2}^\top V y) \geq \dots (x_{t_{|x|}} U_{t_{|x|}}^\top V y).$$

For $k = 1$ we obtain the max affinity function.

**RBF** The RBF affinity only weights examples, not individual input and label features following eq. (3):

$$G_{RBF}(x, y) = \sum_{i=1}^{m} \exp(-\lambda_x ||x - \mathbf{x}_i||^2) \exp(-\lambda_y ||y - \mathbf{y}_i||^2)$$

where $\mathbf{x}$ and $\mathbf{y}$ are the sets of vectors from the training set, and $\lambda_x$ and $\lambda_y$ are hyperparameters.

**Latent RBF** The RBF affinity may perform poorly when the input features are too sparse. One way to improve it is to first map the inputs and/or labels into a lower dimensional latent space, and then compute the RBF there instead:

$$G(x, y) = \tag{8}$$
$$\sum_{i=1}^{m} \exp(-\lambda_x ||Ux - U\mathbf{x}_i||^2) \exp(-\lambda_y ||Vy - V\mathbf{y}_i||^2)$$

Again, this requires that the parameters $U$ and $V$ be learnt appropriately for good performance.

**Hybrid Latent RBF** In the case of a finite, fixed set of labels, particularly if the set is well covered by the training data, it may be better to consider a hybrid of the RBF and latent RBF:

$$G(x, y) = \sum_{i=1}^{m} \exp(-\lambda_x ||Ux - U\mathbf{x}_i||^2) \exp(-\lambda_y ||y - \mathbf{y_i}||^2)$$

Here, a latent space is used for comparison of input $x$ with the input part of the training data, but for the labels the original space is used. That is because we may not have a problem with sparsity in the label space so the latent space there is no longer needed. In the extreme case of very large $\lambda_y$ we instead get:

$$G(x, y) = \sum_{i=1}^{m} \exp(-\lambda_x ||Ux - U\mathbf{x}_i||^2)[y = \mathbf{y_i}]$$

i.e, we only sum over the subset of the training data which have the label $y$ (the parts of the sum where the labels $y$ and $y_i$ are equal). This is useful in the label annotation or item ranking settings where a label from a finite fixed set is either selected or not, but would not be a good idea in an information retrieval setting where a test document may not match any of the training documents.

**$k$-NN Affinity** Instead of computing a smooth $G$ as above we can clip (sparsify) $G$ by taking only the top $k$ nearest neighbors to $Ux$, and set the rest to 0:

$$G(x, y) = \sum_{j \leq k} \exp(-\lambda_x ||Ux - U\mathbf{x}_{t_j}||^2)[y = \mathbf{y_{t_j}}] \tag{9}$$

where

$$||Ux - U\mathbf{x}_{t_1}||^2 \leq ||Ux - U\mathbf{x}_{t_2}||^2 \leq \dots ||Ux - U\mathbf{x}_{t_{|\mathcal{X}|}}||^2$$

So, for each training example, we simply have to find the $k$ nearest neighboring examples in the embedding space, and then we reweight their labels using eq. (9). This means for any training point only at most $k$ labels have a non-zero score, so if $U$ is known it makes it feasible to compute all the neighbors in advance and store in memory or on disk, even when the number of labels is very large. We use this trick to speed up training via an iterative method, described in section 3.3.

**Approximate $k$-NN Affinity** As computing nearest neighbors can be slow one might want to consider approximate methods for speeding that part up. Many algorithms have been developed for this goal and typically rely on either hashing in the input space e.g. via locality-sensitive hashing (LSH) (Indyk & Motwani, 1998) or through building a tree over the inputs (Bentley, 1975). Applying one of those approaches to find the indices of the nearest neighbors $t_1, \dots, t_k$, then computing eq. (9) would result in a much faster method.

**Cluster Affinity** Finally, in the same spirit of providing an affinity function that is fast to compute, one could simply use the following:

$$G(x, y) = \begin{cases} 1, & \text{if Ux and y are in the same cluster,} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

First, one uses a clustering algorithm of choice to hierarchically cluster the embedded examples $U\mathbf{x}_i$, $i = 1, \dots, m$ (e.g. hierarchical $k$-means). Then, one keeps the top $N$ most frequently occuring labels in each leaf node. Inference time with such an affinity function is now actually faster than even a linear embedding model. That is because a linear embedding model has to score all labels, but eq. (2) using (10) only has to score the labels that are in the same cluster as the input.

While in this paper we do not perform any experiments using this affinity function, we note that the work of (Makadia et al., 2013) already developed exactly this approach, without seeing it within our general framework. They reported large speedups (from 30-1000x faster) with no loss in precision, and sometimes with improved precision as well due to the nonlinear nature of the affinity.

**Algorithm 1** Affinity Weighted Embedding SGD training.

```
Initialize model parameters $U$ and $V$ (we use mean 0, standard deviation $\frac{1}{\sqrt{m}}$.
repeat
    Pick a random example input $x$.
    For example $x$ randomly pick a positive item $x \in \mathcal{Y}_x$.
    Compute $f(x, y)$, defined in eq. (2).
    Set $N = 0$.
    repeat
        Pick a random item $\bar{y} \in \mathcal{Y} \setminus \mathcal{Y}_x$.
        $N = N + 1$.
    until $f(x, \bar{y}) > f(x, y) - \gamma$ or $N \geq |\mathcal{Y} \setminus \mathcal{Y}_x|$
    if $f(u, \bar{d}) > f(u, d) - \gamma$ then
        Make a gradient step to minimize:
            $L\big(\frac{|\mathcal{Y} \setminus \mathcal{Y}_x|}{N}\big) \max(0, \gamma + f(x, \bar{y}) - f(x, y))$.
        Project weights to enforce constraints, e.g. if $||V_i|| > 1$ then set $V_i \leftarrow V_i/||V_i||$ (and similar for $U$).
    end if
until validation error does not improve.
```

## 3.2. Training Objective

Now that we have described the model, the next step is to describe how to train it. We could train such a model using any standard objective, for example a regression (least squares) approach such as in SVD, but in this work we focus on learning to rank as it has previously been observed to perform well for linear embedding models on a number of tasks (Shi et al., 2012; Weston et al., 2011; 2013). Note that the ranking objective is a supervised objective and requires training data consisting of positive input-output examples which we denote $\mathbf{x}_i$, $i = 1, \ldots, m$ and $\mathcal{Y}_{\mathbf{x_i}}$, $i = 1, \ldots, m$. Here $\mathcal{Y}_x$ denotes the set of positives labels for a given input $x$. For example, in a recommendation task this could be the set of items that the user has purchased / watched / listened to, depending on the context. If each training example has only one possible relevant label/item/document it can be more convenient to write $\mathbf{x}_i$, $\mathbf{y}_i$, $i = 1, \ldots, m$ instead.

Our starting point is the objective of the linear embedding model, WSABIE (Weston et al., 2011), which learns the model parameters by minimizing:

$$\sum_{i=1}^{m} \sum_{y \in \mathcal{Y}_{\mathbf{x}_i}} \sum_{\bar{y} \notin \mathcal{Y}_{\mathbf{x}_i}} L\big(rank_y(x)\big) \max(0, \gamma + f(x, \bar{y}) - f(x, y)), \tag{11}$$

where $\gamma$ is the margin, a hyperparameter one must choose.

In the absence of negative data, the above objective tries to rank all the positive labels as highly as possible. Here, $rank_y(x)$ is the rank of the positive label $y$ relative to all the negative labels:

$$rank_y(x) = \sum_{\bar{y} \notin \mathcal{Y}_x} I(f(x, y) \leq \gamma + f(x, \bar{y})),$$

and $L(\eta)$ converts the rank to a weight. Choosing $L(\eta) = C\eta$ for any positive constant $C$ optimizes the mean rank, whereas a weighting such as $L(\eta) = \sum_{i=1}^{\eta} 1/i$ optimizes the top of the ranked list, as described in (Usunier et al., 2009). To train with such an objective, stochastic gradient descent (SGD) has previously been employed, i.e. during each SGD step one samples a triple $(x, y, \bar{y})$ from the sum in eq. (11). For speed the computation of $rank_y(x)$ is then replaced with a sampled approximation: sample $N$ items $\bar{y}$ until a violation is found, i.e. $\max(0, \gamma + f(x, \bar{y}) - f(x, y))) > 0$ and then approximate the rank with $|\mathcal{Y} \setminus \mathcal{Y}_x|/N$. At each step, one typically also enforces that $||U_i|| \leq 1$ and $||V_j|| \leq 1$, for all $i$ and $j$, as a means of regularization.

For a fixed affinity function $G$ (i.e., which we are not learning) we can use almost the same optimization procedure as used for the linear embedding model in (Weston et al., 2011), we just need to compute the relevant gradients for our model. Pseudocode is given in Algorithm 1. In the next section we discuss training for the case of learning $G$ as well.

## 3.3. Learning $G$

Here, for the case of a parameterized $G$, we describe two ways to learn the parameters of $G$ as well as the model parameters $U$ and $V$: either jointly or via an iterative approach.

**Joint learning** The most straight-forward method is simply to perform SGD as in Algorithm 1, but update the parameters of $G$ as well during the gradient update step. This works well for some choices of $G$, such as the sigmoid or top-$k$ affinities, but if they are costly to compute such as the $k$-NN affinity (cf. eq. (9)) it can be very slow. For that reason we explore an iterative solution instead for those cases.

**Iterative learning** While it may be possible to learn the parameters of $G$ jointly with $U$ and $V$ an iterative approach is also possible:

1. Train a standard embedding model: $f(x, y) = x^\top U^\top V y$ as an initialization step for $U$ and $V$.
2. Train $G$, fixing $U$ and $V$.
3. Train $U$ and $V$, fixing $G$.
4. Possibly repeat steps 2-3, or else stop early.

For affinity functions employing the parameters $U$ and $V$ (see subsection 3.1) step 2 just requires $G$ to be built using the embedding $U$ learnt in step (1) (i.e. no actual optimization), and that is then used to build a new embedding model in step (3). Due to the iterative nature of the steps we can compute $G$ for all examples in parallel using a MapReduce framework, and store the training set necessary for step (3), thus making learning efficient. This procedure works well e.g. for the $k$-NN affinity.

## 4. Relation to Previous Work

Some of the simplest forms of machine learning are linear models, e.g. linear SVMs (Vapnik, 1998). They are easy to understand and implement and engineers can encode knowledge via features in a straight-forward fashion. However, their capacity is fixed so performance will saturate as training sizes increase. Moreover, these models can take too much memory for large scale tasks. e.g. for hundreds of thousands of labels (unless sparsity is skillfully employed).

Linear embedding models are almost as simple, having similar properties to linear models, but can take much less memory. This is because they effectively factorize the model parameters rather than storing the full matrix. We already detailed many examples of these methods in section 2.

It is hard for embedding models to overfit because of the low dimension used, and because they are effectively sharing weights between all classes by sharing the embedding space (a kind of multi-tasking) in contrast to e.g. one-vs-rest SVMs. For example, this may be advantageous in the case where there are very few examples for some classes and many examples for others. However, these models can underfit - their capacity is small due to the linear mapping into a low dimensional embedding space, and saturates on large scale datasets. There is an inability to memorize certain feature-class pair relationships due to the low rank of the matrix $U^\top V$. For example, in an information retrieval setting if one wants to model whether input $x$ and label $y$ share the same bag of words, this is not exactly possible because that would require a full rank matrix, i.e. $f(x, y) = x^\top I y$, but we only have $f(x, y) = x^\top W y$, where $W = U^\top V$. Similarly in recommendation they cannot memorize the user-item matrix. Some solutions to this problem have been explored in (Bai et al., 2010). In general, semantically similar objects will obtain close-by vectors in the embedding space, which leads to good generalization properties, e.g. word synonyms are close. However, this can also be a curse when exact differentiation between similar things is required, e.g. the difference between Monday and Tuesday may be important for some queries, but they will likely have similar vectors[1]. AWE fixes some of the above problems because the memorization ability lost in embedding models can be encoded in the affinity function $G$. Moreover, AWE also retains the desirable generalization properties of the embedding vectors.

Of course, there exist many other ways of building nonlinearities in maching learning methods. There are several neural network architectures that can be seen as using embeddings followed by nonlinearities. For example, the neural network language model of Bengio et al. (2003) learns an embedding vector for each word and then nonlinearly combines them to predict the next word to occur in a sentence. Since then, other works have explored further text embedding applications, e.g. (Collobert et al., 2011). In a more general setting, siamese network approaches learn a nonlinear embedding for matching two items, they are often applied to images (Salakhutdinov & Hinton, 2007; Hadsell et al., 2006).

Nearest neighbor is another way achieving nonlinearity by way of memorization, although generalization may be lacking due to the local nature of the decision. Kernel methods, e.g. with an RBF kernel, can be see as trying to fix this problem by learning the global weighting of the RBF centers. However, other kernels can be chosen to encode various prior knowledge as well, see (Shawe-Taylor & Cristianini, 2004) for a review. As previously discussed, in that regard affinity functions have some commonality with kernels and kernel methods.

## 5. Experiments

We conducted experiments on three different tasks: (i) Magnatagatune (annotating music with text tags); (ii) ImageNet (annotation images with labels); and (iii) YouTube video recommendation (recommending videos for a given user). WSABIE has been applied to all three tasks previously (Weston et al., 2012; 2011; Makadia et al., 2013; Weston et al., 2013).

### 5.1. MagnaTagATune

The Magnatagatune dataset requires one to annotate music with text tags (Law et al., 2009). Each example consists of the audio features of the song as input, and a set of one or more tags as the labels. There are 16,289 data examples used for training and validation, 6498 examples used for test, and 160 possible tags. Performance is measured using precision at 1 and 3.

To represent the audio as feature vectors we used MFCC features for both WSABIE and our method, similar to those used in (Weston et al., 2012). For both models we used an

[1]Example given in Percy Liang's invited oral at ICML 2013.

embedding dimension of 100. We optimized the other hyperparameters of WSABIE (learning rate and margin) and then used the same hyperparameters for AWE. We used the latent $k$-NN affinity function (cf. eq. (9)), with $k = 20$, and optimized with the iterative training method, using only 3 steps, Due to the choice of $G$, it makes sense to compare our approach to standard $k$-NN (reoptimizing $k$ in that case), and consider it as a baseline. Finally, as we use a latent $k$-NN affinity we also compare to standard $k$-NN operating in the embedding space induced by WSABIE, i.e. first mapping the inputs $x$ to the embedding space using the learnt matrix $U$, and then computing the neighbors as usual.

The results are reported in Table 1. Our method improved over WSABIE slightly, and was superior to the $k$-NN baselines. The latter shows that the improvement does not come directly from the affinity function alone but rather from the way the model uses that affinity function to improve the learnt embeddings. However, $k$-NN in the embedding space was superior to $k$-NN in the original space, showing the power of the embedding representation. We speculate that the improvement of AWE over WSABIE is only small due to the small size of the dataset (only 16,000 training examples, and 104 input dimensions for the MFCCs). We believe our method will be more useful on larger tasks.

*Table 1.* Magnatagatune Results

| Algorithm | Prec@1 | Prec@3 |
|---|---|---|
| $k$-Nearest Neighbor | 39.4% | 28.6% |
| $k$-NN (Embedding space) | 45.2% | 31.9% |
| WSABIE | 48.7% | 37.5% |
| Affinity Weighted Embedding | 52.7% | 39.2% |

## 5.2. ImageNet

ImageNet (Deng et al., 2009) is a large scale image dataset organized according to WordNet (Fellbaum, 1998). Concepts in WordNet, described by multiple words or word phrases, are hierarchically organized. Mechanical Turk is employed to attach quality-controlled human-verified images to these concepts, and the dataset is growing as more images are reliably labeled.

We used the Fall 2011 version, which contains about 10M images, from which we kept 10% for validation, 10% for test, and the remaining 80% for training. There are around 21,000 possible labels that can be attached to an image which range from animals ("white admiral butterfly") to objects ("refracting telescope"). Our task is, for a given test image, to rank the labels.

We used a similar feature representation as in (Weston et al., 2011) which combines multiple feature representations which are the concatenation of various spatial and multiscale color and texton histograms for a total of about $5 \times 10^5$ dimensions. Then, KPCA is performed (Schoelkopf et al., 1999) on the combined feature representation using the intersection kernel (Barla et al., 2003) to produce a 474 dimensional input vector. The same input features are used for both linear embedding (WSABIE) and our method, AWE. We used an embedding dimension of 128 for both.

In (Weston et al., 2011) it was shown that the ranking algorithm WSABIE performs well on this task. WSABIE was shown to be superior to unbalanced one-vs-rest (see also (Perronnin et al., 2012)), PAMIR (Grangier & Bengio, 2008), and $k$-nearest neighbors. Note that since that result deep neural networks (Krizhevsky et al., 2012; Dean et al., 2012) have been shown to be state-of-the-art on this dataset, outperforming other methods by a significant margin.

We used a similar setup for AWE as in the MagnaTagATune dataset: we use the same hyperparameters as for WSABIE and we employ the latent $k$-NN affinity function (cf. eq. (9)), with $k = 20$. Again we use $k$-NN operating in the embedding space as a baseline.

The results are reported in Table 2. On this task we obtained large improvements over WSABIE. Indeed, our method now appears competitive with the convolutional neural network model of (Dean et al., 2012). Note, that this method was run on a different train/test split, although that is unlikely to be a large factor due to the large sizes of the datasets. However, we believe the method of (Krizhevsky et al., 2012) would likely perform better again if applied in the same setting, but the experiments are in a different setup (they use the Fall 2009 version with 10k classes).

*Table 2.* ImageNet Results (Fall 2011, 21k labels). The result marked with an asterisk is on a different train/test split.

| Algorithm | Prec@1 |
|---|---|
| WSABIE (KPCA features) | 9.2% |
| $k$-Nearest Neighbor (WSABIE space) | 13.7% |
| Affinity Weighted Embedding | 16.4% |
| Convolutional Net (Dean et al., 2012) | 15.6%* |

## 5.3. YouTube Recommendations

We next considered another very large scale problem, that of recommending videos from a large online video community, www.youtube.com. The dataset consists of a large set of anonymized users, where for each user there is a set of associated items based on their watch/listen history as well as additional features that indicate their preferences.

The user-item matrix is a sparse binary matrix. The million most popular videos are considered as the set of items (labels) to rank, and our aim is to rank these videos for a given user, to suggest videos that are relevant to the user. The training data is thus of the form where each training pair is based on an anonymized user. For each user the input $x_i$ is a sparse vector of videos that the user has watched. The dataset consists of 100s of millions of examples (users). We set aside 0.5M examples for validation, and 1M for test.

*Table 3.* YouTube Results

| Algorithm | Prec@1 | Prec@10 |
|---|---|---|
| SVD | -54% | -57% |
| WSABIE | - | - |
| AWE $G_{sig}$ $\beta = -1$ | +2% | +12% |
| AWE $G_{sig}$ $\beta = -0.5$ | +9% | +16% |
| AWE $G_{sig}$ $\beta = 0$ | +31% | +29% |
| AWE $G_{sig}$ $\beta = 0.25$ | +51% | +40% |
| AWE $G_{sig}$ $\beta = 0.5$ | +28% | +20% |
| AWE $G_{top-k}$ $k = 1$ | +6% | +5% |
| AWE $G_{top-k}$ $k = 5$ | +18% | +17% |
| AWE $G_{top-k}$ $k = 7$ | +19% | +19% |
| AWE $G_{top-k}$ $k = 11$ | +21% | +21% |

To construct evaluation data, we randomly selected 5 watched items for testing per user, and kept them apart from training. At prediction time for the set of test users we then ranked all test items (i.e. items that they have not watched/listened to that are present in the training set) and observe where the 5 watched test items appear in the ranked list of recommendations. We then evaluate precision at 1 and 10 metrics.

WSABIE has previously been shown to perform well on this problem, including being employed in live experiments on `www.youtube.com` (Makadia et al., 2013; Weston et al., 2013). We took the best performing existing WSABIE system (although for memory and speed reasons we fixed the embedding dimension to be $m = 64$) and used the same parameters for AWE. For AWE we tried two different affinity functions, the max/top-$k$ affinity (cf. eq. (7)) and the linear sigmoid (cf. eq. (6)). We trained using joint learning.

As well as comparing to the natural baseline WSABIE, we also compare to SVD, a popular method for recommendation tasks. For SVD we use the 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.

Results are given in Table 3, we report relative changes in the metrics compared to the WSABIE baseline. Firstly, the WSABIE baseline outperforms SVD, as has been observed in previous work (Weston et al., 2013) presumably because it is better at optimizing the evaluated ranking metrics due to its loss function which (approximately) optimizes them.

We then report results of AWE with the two affinity functions $G_{sig}$ and $G_{top-k}$ with varying choices of their hyperparameters. We observe improvements across a wide-range of these parameters. The smallest gains are for $G_{top-k}$ for $k = 1$ which uses only the max scoring feature (c.f. eq. (7)), perhaps because choosing only the max is too brittle and ignores too many features, and for $G_{sig}$ for $\beta = -1$ (in all cases $\alpha = 1$). The largest gains are a 51% improvement in precision@1 and a 40% improvement in precision@10 using $G_{sig}$ with $\beta = 0.25$. For both affinity functions, it appears that focusing on the features which are more related on a per-label basis (via the affinity weighting) gives improvements. One can see this as a per-example and per-label form of feature selection. For example, in the YouTube case it means that for a particular music video one might focus on the music videos a particular user has viewed (or, depending on the hyperparameter even more focused such as only the videos in the same music genre) whereas for a comedy video it might focus only on other comedies, and so forth.

## 6. Conclusions

In conclusion, we proposed a class of models: Affinity Weighted Embedding (AWE). By incorporating an affinity function $G$ into supervised linear embedding we obtain a flexible approach for encoding knowledge. Depending how it is defined, $G$ can provide a good blend of memorization and generalization, as well as potentially faster inference. We gave several examples how such an approach with different specific choices of $G$ can benefit real-world applications.

Future work should answer several questions. Firstly, what other functions $G$ are there that are interesting and useful? Is there a choice that gives both improved generalization and speed without prohibitive memory consumption, i.e. so that it has no drawbacks at all?

For some choices of $G$ we have explored, e.g. via $k$-NN affinity, the cost of increased capacity (and improved performance) from using $G$ is that it both increases the storage and computational requirements compared to linear embedding models. For example, one might have to compute the nearest-neighbors, although approximations and/or parallel computing can be employed. However, most nonlinear models, for example SVMs, neural networks or nearest neighbor have this problem as well. One avenue to explore in that regard is to use approximate methods in order to compute $G$.

## References

Bai, Bing, Weston, Jason, Grangier, David, Collobert, Ronan, Sadamasa, Kunihiko, Qi, Yanjun, Chapelle, Olivier, and Weinberger, Kilian. Learning to rank with (a lot of) word features. *Information retrieval*, 13(3):291–314, 2010.

Barla, A., Odone, F., and Verri, A. Histogram intersection kernel for image classification. *Intl. Conf. Image Processing (ICIP)*, 3:III–513–16 vol.2, 2003.

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pierre, and Jauvin, Christian. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):517, 1975.

Billsus, D. and Pazzani, M.J. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 54, pp. 48, 1998.

Blei, David M, Ng, Andrew Y, and Jordan, Michael I. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Chapelle, Olivier, Vapnik, Vladimir, Bousquet, Olivier, and Mukherjee, Sayan. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1-3):131–159, 2002.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Mickael, Kavukcuoglu, Koray, and Kuksa, Pavel ˙ Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Cortes, Corinna, Mohri, Mehryar, and Rostamizadeh, Afshin. Learning non-linear combinations of kernels. In *Advances in Neural Information Processing Systems*, pp. 396–404, 2009.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q., Mao, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, pp. 1232–1240, 2012.

Deerwester, Scott, Dumais, Susan T., Furnas, George W., Landauer, Thomas K., and Harshman, Richard. Indexing by latent semantic analysis. *JASIS*, 1990.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conf. Computer Vision Pattern Recognition (CVPR)*, 2009.

Fellbaum, Christiane (ed.). *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

Grangier, D. and Bengio, S. A discriminative kernel-based model to rank images from text queries. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30:1371–1384, 2008.

Hadsell, Raia, Chopra, Sumit, and LeCun, Yann. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pp. 1735–1742. IEEE, 2006.

Hofmann, Thomas. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 50–57. ACM, 1999.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613. ACM, 1998.

Koren, Yehuda, Bell, Robert, and Volinsky, Chris. Matrix factorization techniques for recommender systems. *Computer*, 42 (8):30–37, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

Law, Edith, West, Kris, Mandel, Michael I, Bay, Mert, and Downie, J Stephen. Evaluation of algorithms using games: The case of music tagging. In *ISMIR*, pp. 387–392, 2009.

Lee, D.D. and Seung, H.S. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2001.

Makadia, A, Weston, J, and Yee, H. Label partitioning for sublinear ranking. In *International Conference on Machine Learning, ICML, Atlanta*, 2013.

Perronnin, F., Akata, Z., Harchaoui, Z., and Schmid, C. Towards good practice in large-scale learning for image classification. In *CVPR*, 2012.

Salakhutdinov, Ruslan and Hinton, Geoffrey E. Learning a nonlinear embedding by preserving class neighbourhood structure. In *International Conference on Artificial Intelligence and Statistics*, pp. 412–419, 2007.

Schoelkopf, B., Smola, A. J., and Müller, K. R. Kernel principal component analysis. *Advances in kernel methods: support vector learning*, pp. 327–352, 1999.

Shawe-Taylor, John and Cristianini, Nello. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Shi, Yue, Karatzoglou, Alexandros, Baltrunas, Linas, Larson, Martha, Oliver, Nuria, and Hanjalic, Alan. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pp. 139–146. ACM, 2012.

Usunier, N., Buffoni, D., and Gallinari, P. Ranking with ordered weighted pairwise classification. In *ICML*, 2009.

Vapnik, V. *Statistical Learning Theory*. Springer, 1998.

Weston, J., Bengio, S., and Usunier, N. Wsabie: Scaling up to large vocabulary image annotation. In *Intl. Joint Conf. Artificial Intelligence, (IJCAI)*, pp. 2764–2770, 2011.

Weston, J., Bengio, S., and Hamel, P. Large-scale music annotation and retrieval: Learning to rank in joint semantic spaces. In *Journal of New Music Research*, 2012.

Weston, Jason, Yee, Hector, and Weiss, Ron J. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM conference on Recommender systems*, pp. 245–248. ACM, 2013.