Fakultät für Informatik

der Technischen Universität München

Master's Thesis in Informatik

Image Object and Document Classification using Neural Networks with Replicated Softmax Input Layers

Jörg Landthaler

Fakultät für Informatik
der Technischen Universität München

Master's Thesis in Informatik

Image Object and Document Classification using Neural Networks with Replicated Softmax Input Layers

Bildobjekt- und Dokumentklassifizierung mittels Neuronaler Netzwerke mit Replicated Softmax Eingabeschichten

Jörg Landthaler, B.Sc.

Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dipl.-Inf. Christian Osendorfer
Submission Date:	21st of December 2011

I assure the single handed composition of this master's thesis only supported by declared resources.

Garching, 21st of December 2011, Jörg Landthaler

Abstract

This thesis explores a variant of the bag-of-visual-words framework with a large fraction of unsupervised learning to predict the presence or absence of objects in images. We extract local image features with different SIFT detector and descriptor implementations from the PASCAL VOC 2007 dataset. Based on the bag-of-visual-words assumption, we quantize the visual words into visual word counts using Sculley's Mini-batch k-Means. Afterwards, we train Neural Networks with Replicated Softmax input and multilabel classification output layers. We use these Neural Networks with multiclass classification output layers (softmax) for the task of document classification, too.

Major contributions of this thesis encompass a detailed mathematical derivation and implementation of the Replicated Softmax (RSM) model presented by Salakhutdinov and Hinton, as well as a detailed mathematical derivation of Welling et al.'s Exponential Family Harmoniums (EFH). We can report classification results on the 20 Newsgroups dataset competitive e.g. to the directed DiscLDA model. Moreover, Neural Networks with RSM input layers significantly outperform standard Feed-forward Neural Networks on the PASCAL VOC 2007 image object classification challenge (in terms of mean Average Precision).

Moreover, we present the DualRSM model for image object classification that adds a second visible units wing to the RSM model and hence enables it to combine two different histogram data inputs. In particular, we train it on visual word counts together with their respective all-to-all distances histogram. This is an attempt to incorporate information about the spatial relationships among the visual words, i.e. to leverage the strongly simplifying bag-of(-visual)-words assumption.

Kurzfassung

Diese Arbeit untersucht eine Variante des 'bag-of-visual-words' Frameworks mit einem großen Anteil unüberwachten Lernens, um die Anwesenheit oder Abwesenheit von Objekten in Bildern vorherzusagen. Wir extrahieren Bildworte mit verschiedenen SIFT Detektoren und Deskriptoren aus den Bildern des PASCAL VOC 2007 Datensatzes und transformieren die Bildworte in Bildworthäufigkeiten unter Verwendung des Mini-batch k-Means Clustering Algorithmus. Dabei wird vereinfachend angenommen, dass Bildworten, analog zu Worten in Dokumenten, auch ohne Beachtung der Auftretensreihenfolge Aussagekraft über das entsprechende Bild bzw. Dokument innewohnt. Anschließend trainieren wir künstliche Neuronale Netze mit Replicated Softmax Eingabeschichten und mehreren binären Ausgabeneuronen in der Ausgabeschicht auf den Bildworthäufigkeiten, sowie auf Worthäufigkeiten extrahiert aus dem 20 Newsgroups Datensatz.

Zentrale Beiträge dieser Arbeit umfassen eine detaillierte mathematische Herleitung und Implementierung des Replicated Softmax (RSM) Modells von Salakhutdinov und Hinton, sowie eine detaillierte mathematische Herleitung von Welling et al's Exponential Family Harmonium (EFH) Modell. Wir können Klassifizierungsergebnisse vorlegen, die auf dem 20 Newsgroups Datensatz konkurrenzfähig zu dem gerichteten DiscLDA Modell sind. Darüber hinaus lassen die Klassifierungsergebnisse von Neuronalen Netzen mit Replicated Softmax Eingabeschichten auf der PASCAL VOC 2007 Klassifizierungsaufgabe normale vorwärtsgerichtete Neuronale Netze hinter sich.

Außerdem präsentieren wir das DualRSM Modell zur Bildobjektklassifizierung, welches die Eingabeschicht des RSM Modell dahingehend erweitert, dass zwei verschiedene Typen von Eingabedaten eingesetzt werden können, insbesondere Bildworthäufigkeiten in Kombination mit Abstandshäufigkeiten der Bildworte. Somit handelt es sich um einen Versuch, die stark vereinfachende Annahme, dass die Reihenfolge von Bildworten vernachlässigbar ist, abzuschwächen, indem Informationen über räumlichen Beziehungen unter den Bildworten miteinbezogen werden.

Acknowledgments

First and foremost, I am particularly thankful to my advisor Christian Osendorfer for giving me the opportunity to do this thesis, who guided me for over a year into the field of Machine Learning and tangent areas and always fed me with fresh, interesting and highly useful advise.
I want to thank Jan Schlüter for a very prosperous email conversation. I also want to thank Tijmen Tieleman and Volodymyr Mnih for their fast and useful help with Gnumpy, as well as Dr. Peter Gehler for quickly answering my question about the Rate Adapting Poisson paper.
Last but not least, I am deeply thankful to my parents for their patience and support throughout my studies.

Contents

1 Introduction

Enabling machines to understand images has been and still is a field experiencing strong interest. Robots for example heavily depend upon detection and classification of objects. They have to weld the metal rather than to endanger human body parts. An interesting domain of image interpretation constitutes picture search in the World Wide Web. One approach to this objective is to annotate images manually. However, it is desirable to abandon the imperative of annotations, put another way to automate this process. Google recently launched an application capable of both: Searching for similar images given one image or searching for images given words as input. In fact, it is even capable of combining both types of input. Possible use cases are vast and diverse: Imagine for example you make a picture of someone wearing a nice shirt and you want to find a webshop where you can buy it, or you take a photo of a person and later on you search for this person's online profile in social communities, to name but a few.

A common approach to infer image content - without resorting to annotation data - is to capture 'visual words' and feed them into an algorithm stemming from the field of document retrieval or information retrieval. The task of document retrieval is to seek by keywords for documents from a corpus (a collection of documents), see e.g. Manning et al. [32] for an introduction. Document retrieval systems are usually based on indexing the corpus. Therefore, they often exploit the simplifying bag-of-words assumption, i.e. each document is represented by an unordered list of corpus-wide keywords. Typically, they are aggregated to word counts (also called term frequency (tf)). Visual words are local descriptors extracted from images and are assumed to act like their document retrieval counterparts. Image retrieval then can be carried out by feeding a document retrieval algorithm with the visual word counts as features, also referred to as bag-of-visual-words framework, Sivic and Zisserman [52] or vector quantization (VQ) approach, e.g. in Bengio et al. [66].

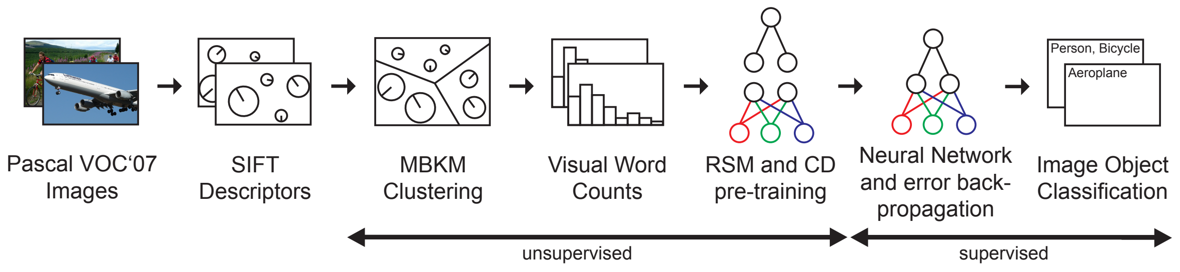

Figure 1.1: Schematic illustration of our image object classification pipeline.

For our goal of image object classification we use a pipeline of algorithms as depicted in Figure 1.1. The visual words are extracted from images contained in the Pascal VOC 2007 dataset [13] by application of the well-known SIFT (Scale-Invariant Feature Transform) [28] algorithm presented by Lowe in 1999. Next, the visual words are quantized into visual word counts using Sculley's Mini-batch k-Means (MBKM) algorithm [51], a computationally fast, memory saving and on the GPU easily parallelizable variant of the classical k-Means clustering algorithm. We determine, if one or more objects of the 20 different classes provided by the dataset are present in an image by feeding a Neural Network with a special input layer. Salakhutdinov and Hinton's stand-alone Replicated Softmax (RSM) model [47] naturally integrates as an input layer of a Neural Network[1]. The Neural Network is conceptually trained as a Deep Belief Network, see Bengio et al. [6], Hinton and Salakhutdinov [17], i.e. pre-trained unsupervised, layer by layer with Hinton's Contrastive Divergence [18] learning and subsequently fine-tuned by error back-propagation. The Neural Networks achieve multilabel image object classification with a sigmoidal output layer, which is pre-trained with single layer error backpropagation.

Besides testing my Replicated Softmax implementation on the 20 Newsgroups dataset using dictionaries based on most frequent words and highest information gain for document retrieval, we also train Neural Networks with Replicated Softmax input and softmax output layers for multiclass document classification on the 20 Newsgroups dataset.

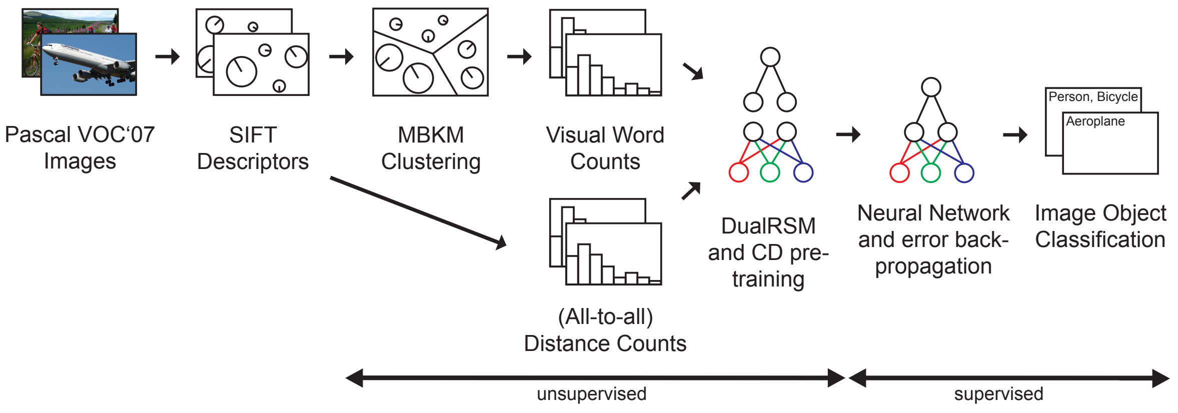
Pascal VOC'07 Images
SIFT Descriptors
MBKM Clustering
Visual Word Counts
(All-to-all) Distance Counts
DualRSM and CD pre-training
Neural Network and error back-propagation
Person, Bicycle
Aeroplane
Image Object Classification
unsupervised
supervised

Figure 1.2: Schematic illustration of our DualRSM image object classification pipeline.

Moreover, we introduce the DualRSM model. It extends the RSM model in the image object classification application, analogous to a Dual Wing Harmonium [62] introduced by Xing et al., to support two different types of count input data. Burghouts et al. [10] report that all-to-all distances of visual words typically obey Weibull distributions. My vision was that these distance counts carry discriminative information about the objects present in images. Therefore, the DualRSM model is fed with both: the visual word counts and the all-to-all visual word distance counts, as illustrated in Figure 1.2. This is an attempt to take the spatial relationships

[1]Do not get confused: We use the term Neural Network here, because Deep Belief Networks are typically defined to have several layers of hidden units. As it turns out, and unfortunately on contrary to our initial intention, more than one layer of hidden units does not lead to better results in our pipeline. Nevertheless, we still pre-train our Neural Networks layer-wise using Contrastive Divergence learning as it is common in Deep Belief Networks.

of the descriptors into account by using the frequencies of the discretized distances among the descriptors, i.e. we attempt to overcome or at least to moderate the bag-of-words assumption. It can also be interpreted as an image retrieval counterpart to Zhang et al.'s [65] term connections frequency for document retrieval.

Our primary focus dwells on Neural Networks with RSM and DualRSM input layers for image object classification. Nevertheless, along the multiple stages of the complete pipeline a variety of different approaches has been proposed. Images can be represented in various fashions. However, in order to apply document retrieval algorithms to the image retrieval problem, utilizing visual words is common practice. There exists a bunch of local feature detectors and descriptors, e.g. GLOH [35], SURF [3], HOG [11], LESH [49], see Mikolajczyk and Schmid [35] for an overview of detectors. Quite some effort was put into reducing the dimensionality of the 128-entries SIFT descriptor vectors e.g. PCA-SIFT by Ke and Sukthankar [24], Winder and Brown [60] or training a Deep Belief Network on the descriptors, in order to apply the vector space model from information retrieval, see Philbin et al. [41]. However, it is rare that the parameters of the SIFT detector are inspected, even though they have crucial impact on the quality of the descriptors localized. We stick with the original SIFT detector and descriptor for reasons of comparability, but compare different SIFT implementations and PCA-SIFT(36). We attempted to obtain more distinctive keypoints by using custom parameters for the SIFT detector.

The next step in the pipeline is to map the visual words to keywords, i.e. to define a dictionary. The classical approach is to apply a clustering algorithm like the k-Means algorithm. Unfortunately, k-Means scales poorly to large corpora. Hierarchical- [38] and Approximate- [40] k-Means are modifications to the original k-Means algorithm to circumvent these shortcomings through local decisions in high-dimensional space, relaxing the quality requirements to the resulting cluster centers on the other hand. Another approach is to express the descriptors as a linear combination of keywords, see e.g. Bengio et al. [4] or Zhou et al. [66], possibly even online, e.g. Mairal et al. [31]. This is usually called sparse coding. They report very good results, but quantizing the SIFT descriptors is not the priority of this thesis. We choose the Mini-batch k-Means algorithm for this step, because it is an unsupervised algorithm and because of its scalability properties.

The last step in the pipeline is to use the keyword counts for classification. Typical classification methods are Support Vector Machines, used e.g. by Bengio et al. [4], Zhou et al. [66], or Neural Networks. We propose to use Neural Networks with special input layers: first a Replicated Softmax layer, naturally modeling word counts as input, and secondly a DualRSM layer, naturally modeling word and distance counts as input.

Other approaches to image object classification are e.g. to feed a Neural Network directly with raw pixel values. This was explored e.g. by Krizhevsky [25], however only on tiny images subdivided into patches with Gaussian-Bernoulli Restricted Boltzmann Machines (RBM). It can be seen as an undirected counterpart to convolutional networks, see e.g. Bishop [7] Subsection 5.5.6.. Roux et al. [44], similarly divide images into patches, learn RBMs on them and combine the results with other types of RBMs. They even employ softmax units, but not the hidden biases rescaling of the RSM model.

Except for the descriptor extraction, all subsequent steps are carried out using minibatch variants of stochastic gradient descent. Therefore, the complete pipeline scales well. As illustrated in Figures 1.1 and 1.2 large fractions of our pipeline are governed by unsupervised learning. Thus, it is possible to run a crawler on the web and to learn reasonable parameters for different stages of the pipeline easily on a large corpus.

This thesis is organized as follows: Chapter 2 introduces the existing and essential prerequisites for our own work. It covers the datasets and fundamental algorithms from all stages: Scale Invariant Feature Transform (SIFT) and Mini-batch k-Means; Logistic Regression and Multiclass Logistic Regression as building blocks of Neural Networks and Restricted Boltzmann Machines as building blocks of Deep Belief Networks. Furthermore, we describe the chain of different RBM/EFH models that lead to the Replicated Softmax model. It also describes Contrastive Divergence learning for pre-training as well as the error backpropagation algorithm for fine-tuning Deep Belief Networks.
Long mathematical derivations are mostly outsourced to Appendix C. Stochastic gradient descent and its minibatch variants as well as the momentum technique are used throughout this work and therefore outsourced to Appendix B.

Chapter 3 covers the work contributed by us. In particular, Section 3.1 shows that my implementation reproduces results for document retrieval from Salakhutdinov and Hinton's paper on the 20 Newsgroups dataset. It also reveals interesting technical details on how to extract most frequent word dictionaries from the 20 Newsgroups dataset plus good image retrieval results using dictionaries based on highest information gain. Details on using the RSM model in Neural Networks using a modified error backpropagation algorithm can be found in Section 3.2. Section 3.3 then uses both types of dictionaries in the multiclass document classification setting, i.e. Neural Networks with RSM input and softmax output layers.
Section 3.4 covers the extraction of visual words and visual word counts from the PASCAL VOC 2007 dataset. Section 3.5 is dedicated to the central application of Neural Networks with RSM input and sigmoidal output layers on the visual word counts. Standard Feed-forward Neural Networks are the baseline against our pipeline using different SIFT implementations, kmeans++ initialization of Mini-batch k-Means and Deep Neural Networks with RSM input layers. Moreover, we tried a linear embedding using RSM models with two and three hidden units. Furthermore, we experimented with Support Vector Machines trained on both: visual word counts and codes obtained by a Deep Auto-Encoder.
Section 3.6 presents all-to-all distance counts extraction and the DualRSM model. Note that minor technical prerequisites are treated in the respective Sections of Chapter 3.

All practical work was performed on the hard- and software configuration described in Appendix A. Finally, Chapter 4 concludes this thesis with a short discussion of contributions and future work.

2 Prerequisites and related work

2.1 Datasets

We use two datasets: The 20 Newsgroups dataset serves to verify our RSM implementation and we perform document retrieval and classification on it. For the task of image object classification we selected the frequently used and challenging Pascal VOC 2007 dataset.

2.1.1 20 Newsgroups

The 20 Newsgroups dataset [1] consists of 18.845 Usenet newsgroup entries split (by date version) into 11314 training and 7531 test records. The records originate in 20 different topics and are almost evenly partitioned:

- comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x
- rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey
- sci.crypt, sci.electronics, sci.med, sci.space
- misc.forsale
- talk.politics.misc, talk.politics.guns, talk.politics.mideast,
- talk.religion.misc, alt.atheism, soc.religion.christian

2.1.2 PASCAL VOC 2007

The PASCAL VOC (Visual Object Classes) 2007 dataset [13] comprises 9963 labeled images containing objects from 20 different categories:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

In contrast to the very evenly partitioned 20 Newsgroups dataset, here the class *person* is predominant with 2008 images picturing persons in the train set, whereas the objects of all other

[1] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

classes have an average of 278.84 instances (test set: 2007 versus 263.47).[2] Note that up to 6 objects can be present in one image. The dataset is split into 5011 training set images (we use the train set and the validation set combined as training set) and 4952 test set images of different sizes, see Table 2.1 for further details. This dataset of real-world photos is very challenging, including various camera angles, strong differences in illumination and contrast and even objects from the same class differ vigorously.

			Minimum	Maximum	Average	Standard Deviation
Train	5011 images, 12606 objects, 7307 targets	Height	96	500	383.631411	62.870358
		Width	127	500	472.703053	57.577160
		Objects	1	6	1.457992	0.652327
Test	4952 images, 12032 objects, 7013 targets	Height	139	500	381.537763	63.080221
		Width	148	500	471.246567	59.811408
		Objects	1	5	1.416195	0.617070
Total	9963 images, 24638 objects, 14320 targets	Height	96	500	382.590786	62.980296
		Width	127	500	471.979123	58.699870
		Objects	1	6	1.437218	0.635360

Table 2.1: PASCAL VOC 2007 dataset statistics.

Several ground truth data for training and test set are provided. We use the labels provided for the Classification Challenge, where the target is to predict whether objects from a given class are present in an test image. Hence, multiple occurrences of objects from one class are treated as one object is present. Moreover, 606 targets in the train set 606 and 619 in the test set are tagged as difficult and therefore not considered in the classification challenge.

2.2 Scale-Invariant Feature Transform (SIFT)

In 2004 David Lowe described a full pipeline of algorithms for object recognition: The **Scale-Invariant Feature Transform (SIFT)** [29]. It includes a key point detector and a key point descriptor besides methods for indexing and matching the keypoints from different images and objects. However, often the term SIFT refers to only the detector and descriptor. We employ Lowe's SIFT implementation[3] and the open SIFT detector and SIFT descriptor implementations by Vedaldi (SIFT++, VLFeat[4]) [56] for the purpose of gathering visual words. This Section depicts SIFT detector and descriptor in further detail and briefly discusses alternative approaches. Several details described here source from Vedaldi's documentation [56].

Using SIFT, an image is described by feature points (also called interest points). The intention of feature points is that they (hopefully) hold relevant information about the content of the image at, or to be more precise around these points, image for example edges or corners of objects.

[2]See `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/dbstats.html` for further details on statistics.

[3]http://www.cs.ubc.ca/ lowe/keypoints/

[4]`http://www.vlfeat.org`

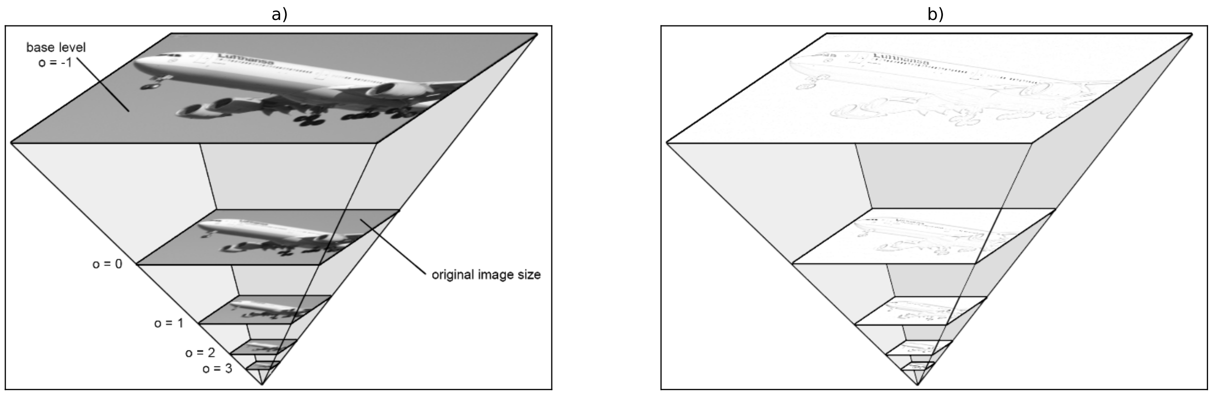

Figure 2.1: a) Gaussian scale space b) Difference of Gaussians scale space.

Both, detector and descriptor are based on the image gradient at different scales. The interest points are detected as local extremes of a Difference of Gaussian scale space and described utilizing the corresponding Gaussian scale space. Within a frame around the detected interest points, a weighted histogram of normalized gradient orientations is calculated and wrapped up in vectors of typically 128 elements. SIFT descriptors are appealing, since they are, to a certain degree, invariant to affine transformations (translation, dilation, rotation) and partially invariant to illumination changes, noise, occlusion and 3D projection. Put another way, one seeks for automatically extracted abstract descriptions of the information contained in images, compared to plain pixel data or manually added information like annotations.

The **scale spaces** can be imagined as flipped image pyramids. First of all, the gray version of the original image $I(x)$ is pre-smoothed using a 1D-Gaussian Kernel (approximated by 7 sample points) in both directions consecutively with $\sigma = 0.5$:

$$G(x, \sigma) = (g(\sigma) \circ I)(x), \quad g(x, \sigma) = \frac{1}{\sqrt{(2\pi)}\sigma} \exp^{\frac{-x^2}{2\sigma^2}} \tag{2.1}$$

Next, the pre-smoothed image is enlarged by factor two in both directions (using bilinear interpolation). This enlarged version of the image serves as the base level in the **Gaussian scale space** pyramid. One proceeds from one level in the pyramid to the next lower level by downsampling by half in each direction, as depicted in Figure 2.1 a). Within each level of the pyramid (also called octave) successively smoothed versions of the resized images are calculated, called sub-levels. Again, the smoothing is achieved by application of the 1D-Gaussian Kernel successively in both directions, cf. Figure 2.2 a) to c).

If the octaves are indexed by o, while the sub-levels are identified by s, then the combination of octave index o and scale index i, i.e. the position in the gaussian scale space, can be characterized through a single parameter σ (scale coordinate):

$$\sigma(o, s) = \sigma_0 2^{o+s/S} \tag{2.2}$$

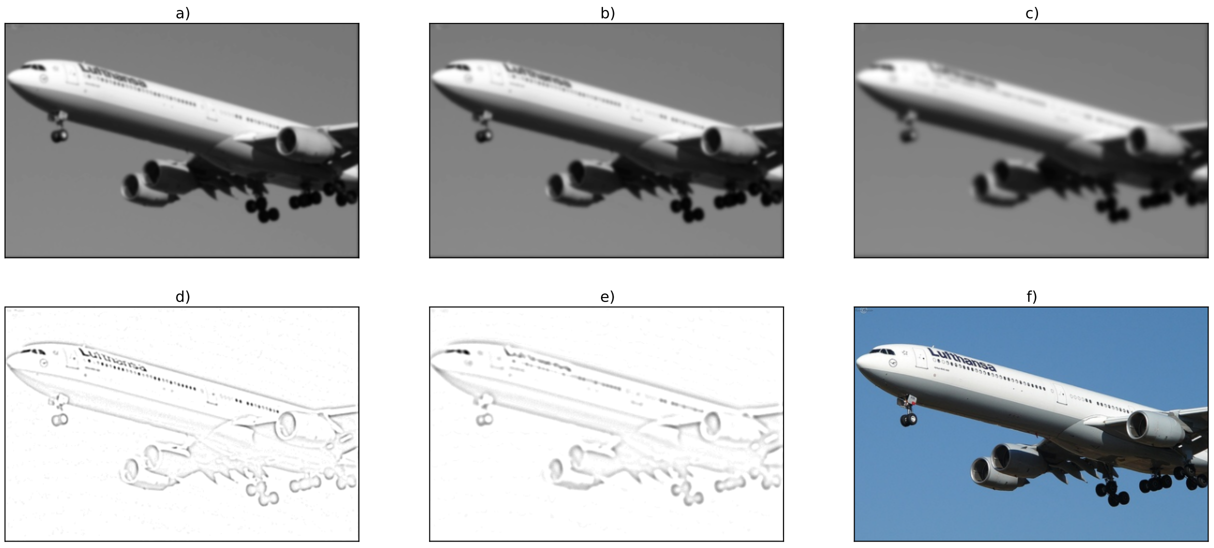

Figure 2.2: Illustration of different Gaussian and Difference of Gaussians sub-levels on level $o = 0$: a) $G(x, \sigma(o = 0, s = 0))$ b) $G(x, \sigma(o = 0, s = 1))$ c) $G(x, \sigma(o = 0, s = 3))$ d) $DoG(x, \sigma(o = 0, s = 0))$ e) $DoG(x, \sigma(o = 0, s = 2))$ f) Original image, from PASCAL VOC 2007 dataset [13].

with $o \in o_{\min} + [0, \ldots, O-1]\,, s \in [0, \ldots, S-1]$, where $o_{\min}$ being the base level (by default -1, which corresponds to the enlarged version of the image by factor 2), O the number of octaves and S the number of sub-levels (by default $S = 3$).

Based on the Gaussian scale space, the **Difference of Gaussians (DoG)** scale space is calculated. The DoG scale space can be thought of as the derivative of the Gaussian scale space along the scale coordinate σ. However, as the name suggest, the DoG can easily be calculated as the difference of two neighboring sub-levels in the Gaussian scale space, see Figure 2.2 d) and e).

$$DoG(x, \sigma(o, s)) = G(x, \sigma(s+1, o)) - G(x, \sigma(s, o)) \tag{2.3}$$

The **SIFT detector** utilizes the Difference of Gaussians scale space. Candidate keypoints are the local extremes of the DoG scale space. Therefore, the algorithm iterates through the levels and sub-levels and checks each pixel against the eight neighboring pixels for a local maximum or minimum. If it is an extremum, then one searches along the scale coordinate for the most intense corresponding 9 pixels. Candidate keypoints are rejected, if the gradient modulus falls below a specific peak threshold. Hence the peak threshold can be elevated such that only strongly peaked keypoints are accepted, cf. Figure 2.3 a) to e). Likewise a candidate keypoint is rejected, if it falls short of a certain edge threshold. The intention is to accept only keypoints on corners rather than in valleys as illustrated in Figure 2.3 f) to i). Due to the up- and downsampling along the levels of the pyramid, the final keypoint spatial coordinates x and y (in the original

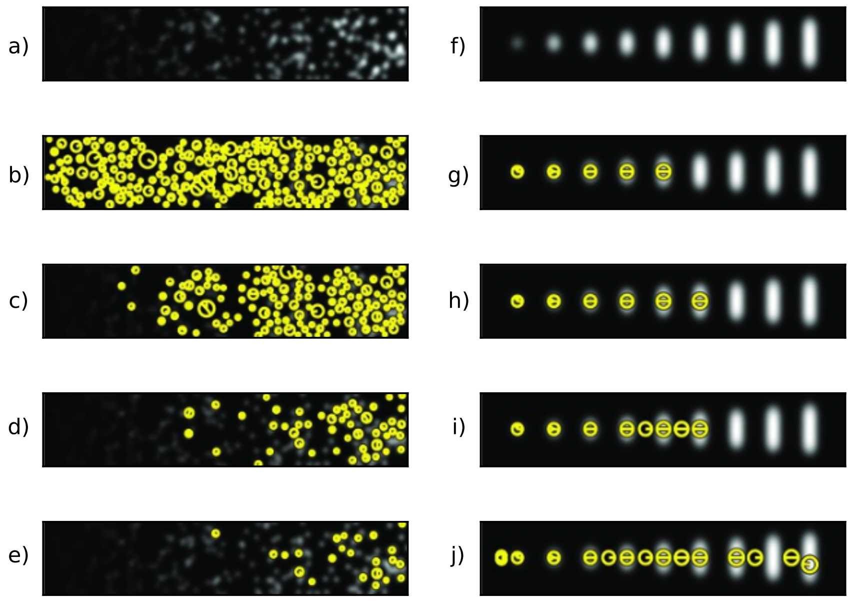

Figure 2.3: Influence of the SIFT parameters peak- and edge-threshold. The center of each yellow circle represents a detected point. a) Test image for peak threshold with values 0, 10, 20 and 30 from b) to e). f) Test image for edge threshold with values 3.5, 5, 7.5 and 10 from f) to h). Images are copied from [57]. See in color for better visualization.

image) are sub-pixel precise (refined using square interpolation).

Moreover, a keypoint comes along with a major orientation, which is calculated as the predominant gradient orientation around the keypoint in a Gaussian Window of deviation $\sigma_w = 1.5\sigma$ (though truncated in reality at $|x| < 4\sigma_w$). The gradient orientations within this Gaussian Window are counted in a histogram. After that the histogram is smoothed by a moving average filter. In the end, the histogram's bin with the maximum value and those bins with a value larger than 80% of the maximum value are identified as major orientations. Hence, several keypoints at the same spatial and scale coordinates, but with different major orientations, can be found. Finally, one keypoint is described by two spatial coordinates, one scale coordinate and one major orientation.

The **SIFT descriptor** provides a compact representation of the gradient orientations around a detected keypoint. Therefore, the descriptor utilizes the keypoint detector information and the Gaussian Scale Space. Imagine a virtual square (called frame or support) around the spatial coordinates of each keypoint rotated to align with the major orientation of the detected keypoint, cf. the enclosing red squares in Figure 2.4 a). Figure 2.4 is generated using Vedaldi's VLFeat. However, be aware that Figure 2.4 a) actually depicts the resulting SIFT descriptors (histograms), whereas according to Vedaldi the visualization in Figure 2.4 b) is used to depict the SIFT frames.

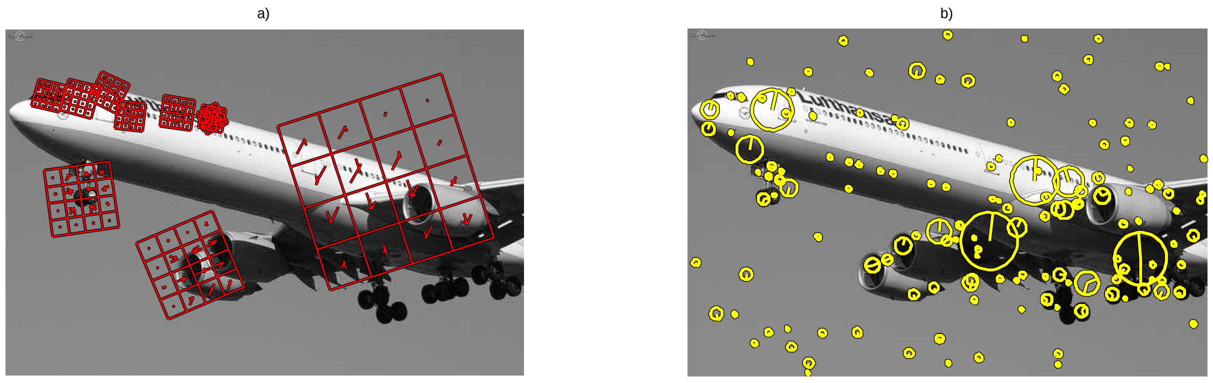

Figure 2.4: Illustration of the SIFT descriptors: a) Selected SIFT frames (support of descriptors), b) All resulting SIFT descriptors.

The descriptors are calculating as follows: The virtual disk is clipped with the actual Gaussian on the detected level and sub-level. I.e. virtual disks close to the border of the Gaussian image might lose some of their area. Next, the pixels in the area of the frame are calculated back, such that they are aligned to the major orientation (using bilinear interpolation).

Now, the square is subdivided into four times four sub-squares. Within each sub-square the gradient orientations are taken into account and discretized into 8 bins, such that they cover full 360°. Each bin then contains the counts of the complying gradient orientations. The gradient orientations in the 4 times 4 spatial bins together with the 8 bins for the orientation build up a three-dimensional histogram with $4*4*8 = 128$ bins. Additionally, the histogram is weighted by the gradient modulus and a Gaussian Window. In the end, the 128-entries vector is normalized (such that value of each entry is between 0 and 255). Each bin then represents the amount of respective weighted gradient orientations. Likewise, each red sub-square of each descriptor in Figure 2.4 a) contains several orientation strokes of different length. Vedaldi's implementation, however, does not necessarily produce the exact same results as Lowe's original implementation. From now on, we use the SIFT frame visualization (yellow in Figure 2.4 b)) to illustrate SIFT descriptors due to their clearer visualization.

Schmid and Mikolajczyk evaluated the performance of different detectors (Harris points, Hessian-Laplace regions, Harris-Affine regions, Harris-Laplace) and descriptors together with the presentation of a new SIFT-alike descriptor: Gradient location-orientation histogram (GLOH). One result is, that SIFT-like descriptors perform best, compared to descriptors based on pixel intensities, spatial-frequency techniques, differential descriptors and other techniques, for example sift-alike PCA-SIFT. They also argue for the superiority of their GLOH descriptor, which basically utilizes a circular support region rather than a quadratic one like SIFT. In spite of that, we stick with SIFT detector and descriptor implementations, because they are open and frequently used.

An interesting variation of SIFT is PCA-SIFT introduced by Ke and Sukthankar [24]. PCA-

SIFT works directly on the gradient values in x- and y-direction in the oriented 39x39 pixel support around a keypoint and applies Principal Component Analysis (PCA) on these gradient values, resulting in a descriptor with e.g. 36 entries rather than 128 compared to original SIFT. However, Zha et al. [64] (see also Ding and He [12]) showed that PCA is a continuous relaxation of discrete k-Means clustering, the ensuing algorithm in our pipeline.

Another interesting property of the SIFT descriptors is that their all-to-all distances usually obey a Weibull distribution as claimed by Burghouts et al. [10].

2.3 Mini-batch k-Means (MBKM)

This Section describes Sculleys's Mini-batch k-Means (MBKM) clustering algorithm [51], a computationally fast, memory-efficient and easily parallelizable variant of the classical, unsupervised k-Means algorithm presented by Lloyd in 1957 (published in 1982) [27]. We use this unsupervised method to quantize the SIFT descriptor vectors of each picture into visual word counts.

The goal of clustering is to group subsets of 'similar' or 'related' elements together. Imagine for example to throw a handful of pebbles onto the street. Afterwards, you will be very likely able to identify groups of pebbles that lie closer together than others, i.e. to identify clusters. In this simple case, the human mind is capable of easily choosing a suitable number of cluster centers. In general, determining the number of cluster centers K is not that obvious. Nevertheless, for now we will assume K to be given. In more than three dimensions the goal of clustering remains the same, however, the problem is not even visualizable so easily any more. Even worse, Clustering, in general, is a NP-hard problem, see Aloise et al. [1] Therefore, the original **k-Means algorithm** attempts to find an approximate solution by applying an iterative procedure.

Formally spoken: Given a normalized dataset with N (D-dimensional) data points $\{x_1, \dots, x_N\}$; $x_n \in \mathbb{R}^D$, the task is to find K cluster centers $c_1, \dots c_K; c_k \in C \subset \mathbb{R}^D$ that minimize the objective function

$$\min E(c) = \frac{1}{2}\sum_{n=1}^{N} \|f(C, x_n) - x_n\|_2^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|c_k - x_n\|_2^2 \tag{2.4}$$

where $f(C, x_n)$ assigns each data point to one cluster center, i.e. defines a 1-of-K coding scheme for each data point. Alternatively, it can be written using indicator variables $r_{nk} \in \{0, 1\}$, cf. Bishop [7] Section 9.1. We consider here the L_2-norm (Euclidean Norm), which is indicated by the 2 in the subscript of the modulus. The factor $\frac{1}{2}$ is added for convenience after calculating the gradient and does not affect the minimum of the objective function. It is essential to normalize the dataset before feeding it to the k-Means algorithms. This ensures that the Euclidean distance function applies comparable weights to each variable.[5]

As Bottou and Bengio showed [9], the k-Means algorithm is an instance of the EM-algorithm (Expectation Maximization). The EM-algorithm is a general framework to optimize objective functions with coupled parameters, i.e. variables for that the function cannot be optimized

[5]For an example see: `http://intelligencemining.blogspot.com/2009/07/data-preprocessing-normalization.html`

independently. The core of the EM-algorithm is to alternate clamping one subset of parameters while optimizing for the remaining parameters.
For the k-Means algorithm, we have to alternate updating the cluster centers and updating the assignment of the data points to the cluster centers. The assignment of the data points to the cluster centers is straightforward: Choose for each data point the closest cluster center according to the Euclidean distance:

$$r_{nk} = \begin{cases} 1 & \text{, if } k = \arg\min_j \|c_j - x_n\|_2^2 \\ 0 & \text{, otherwise} \end{cases} \tag{2.5}$$

Next, the cluster centers are updated. Therefore, calculate the derivative w.r.t. the prototypes for the cluster centers c_k and set it to zero:

$$\frac{\partial E(c)}{\partial c_k} = \frac{\partial}{\partial c_k} \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|c_k - x_n\|_2^2 = \sum_{n=1}^{N} r_{nk}(c_k - x_n) \overset{!}{=} 0 \tag{2.6}$$

Solving for c_k yields:

$$c_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}} = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} x_n \tag{2.7}$$

with $N_k := \sum_{n=1}^{N} r_{nk}$. These two steps are iterated until convergence (neglectable small changes), or for a fixed amount of iterations T. Hence, the time-complexity is $\mathcal{O}(TKN)$ in the latter case. The right hand side of Eq. 2.7 calculates the mean of the assigned data points, the reason for calling the algorithm k-Means.

The classical k-Means algorithm is a batch algorithm, i.e. all data points are involved before the variables are updated. MacQueen [30] derived an **online variant** using **stochastic gradient descent (SGD)** by presenting only *one* data point x_n per update step. Since Eq. 2.6 is of the form of Eq. B.3, MacQeen proposed a sequential update rule of the form of Eq. B.4:

$$c_k^{\text{new}} = c_k^{\text{old}} - \eta_n \left(c_k^{\text{old}} - x_n \right) \tag{2.8}$$

with learning rate η_n. Bottou and Bengio [9] showed that the learning rate is optimal, when $\eta_n = \frac{1}{n_k}$. And n_k is assigned the value one, if the current data vector x_n is assigned to cluster k and zero otherwise.

In 2010, Sculley introduced the **Mini-batch k-Means (MBKM)** algorithm [51]. The simple, yet powerful idea is to use minibatches of randomly chosen data points, instead of single data points as in the SGD variant, i.e. applying Eq. B.5 to the sequential update rule:

$$r_{mk} = \begin{cases} 1 & \text{, if } k = \arg\min_j \|c_j - x_m\|_2^2 \\ 0 & \text{, otherwise} \end{cases} \tag{2.9}$$

and

$$c_k^{\text{new}} = c_k^{\text{old}} - \frac{1}{M_k} \sum_{m=1}^{M} r_{mk} \left(c_k^{\text{old}} - x_n \right) \tag{2.10}$$

with $M \ll N$ chosen by the user and $M_k := \sum_{m=1}^{M} r_{mk}$. However, unlike the general form of Eq. B.5, Sculley provided, based on the results of Bottou and Bengio, a prototype-specific learning rate $\frac{1}{M_k}$. Additionally, the records presented in each minibatch are selected randomly: draw from a discrete uniform distribution.
Sculley argues for the superiority of his algorithm over one record stochastic gradient descent: It leads to qualitatively better results, since minibatches contain less stochastic noise compared to single data points, while the computational costs remain low (time-complexity of $\mathcal{O}(TKM)$, with M chosen by the user).

Due to the iterative nature of the family of k-Means algorithms, they can get stuck in local rather than global optima. On the one side of the coin, the non-determinism of the MBKM algorithm can induce the Mini-batch k-Means algorithm to overcome local optima over time of execution better than SGD or the batch version. On the other side of the coin, the non-determinism does not guarantee that the cluster assignments do not flip any more. Hence a fixed number of iterations must be used as stopping criteria. Additional speedup for MBKM can be achieved by applying second binomial formula [6].

Figure 2.5 illustrates how MBKM proceeds on a simple 2-dimensional dataset. The data points are drawn from 3 Gaussians (with means: (0.323, 0.237), (0.229, 0.7), (0.72, 0.55); à 150, 125, 140 points; 418 points in total and a variance of 0.05 in both directions). Note that the decision boundaries (between two colors in b), d) and f)) are linear. In the multidimensional case the boundaries generalize to simplices.

We also experiment with the **k-means++** seeding algorithm[7], introduced by Arthur and Vassilvitskii [2], for the initialization of the MBKM algorithm. The idea is to choose cluster centers far apart from each other with high probability. The first cluster center is a data point chosen uniformly from the dataset. The remaining cluster centers are selected subsequently using D^2 weighting, as the authors call it. Therefore, the distance from each data point to the closest already chosen cluster center is calculated and each data point x is assigned the probability $\frac{D(x)^2}{\sum_{n=1}^{N} x_n D(x)^2}$ with $D(x)$ being the distance to the closest cluster center. From this probability distribution the next cluster center is drawn. After this seeding procedure, we continue with MBKM. The kmeans++ algorithm provides guarantees for the error ($\mathcal{O}(\log k)$-competitive) and usually reduces the necessary amount of iterations of MBKM. Despite kmeans++ received bad marks in Peterson et al.'s comparison of different k-means seeding methods [39], we apply it, due to its widespread use.

The k-Means algorithm - simple yet popular - experienced various other modifications and specializations. Particularly in the field of feature retrieval, e.g. Hierarchical k-Means (HKM) [38] or Approximate k-Means (AKM) [40] have been proposed. HKM uses trees to start with some

[6] `http://blog.smola.org/post/969195661/in-praise-of-the-second-binomial-formula`
[7] `http://scikit-learn.sourceforge.net/`

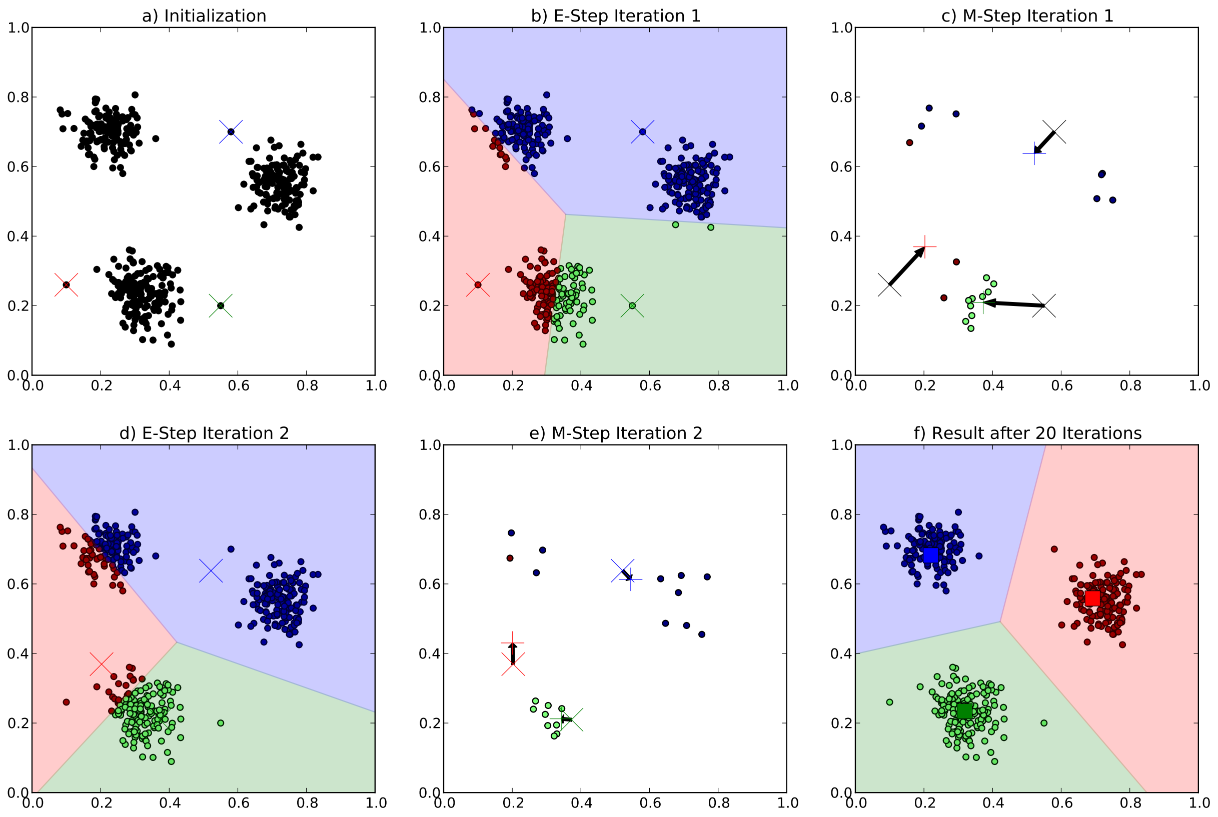

Figure 2.5: Illustration of the Mini-batch k-Means (MBKM) algorithm (K=3, M=20, T=20) on an artificial dataset with data points drawn from three 2-dimensional Gaussians. a) Initialization of the MBKM algorithm by choosing randomly 3 data points from the dataset marked with 'x'. b) Calculate the nearest cluster centers for each data point. Here, the assignments for all data points are shown for clarity - of course, only the assignments for the data points chosen by the minibatch are necessary for the succeeding M-step. c) Use a randomly chosen subset of data points to update the cluster centers by calculating the means of the data points assigned to the cluster centers (indicated by the black arrows; new cluster centers indicated by '+'). Only data points from the randomly chosen minibatch that are used to calculate the update to the cluster centers are displayed. d) Recalculate the nearest cluster centers again. e) Update the cluster centers again. f) The big squares indicate the final cluster centers after 20 full iterations. See in color for better visualization.

initial cluster centers and to recursively subdivide their respective partitions (supports). AKM imposes kd-trees on the cluster centers in each iteration in order to reduce the amount of neighbors and hence the number of distance calculations. They also argue for a better quantization as points that lie close to decision boundaries are treated more robust. At the end of the day, we stick with MBKM, due to its simplicity and its fast and memory-saving execution.

2.4 Logistic Regression & Multiclass Logistic Regression

This Section introduces Logistic Regression and Multiclass Logistic Regression, which are basic discriminative probabilistic classification models. The models are trained with gradient descent or stochastic gradient descent. They serve us as building blocks for Feed-forward Neural Networks. Additionally, we use them as a baseline. This Section tightly follows Bishop [7] Sections 4.3.2. and 4.3.4..[8]

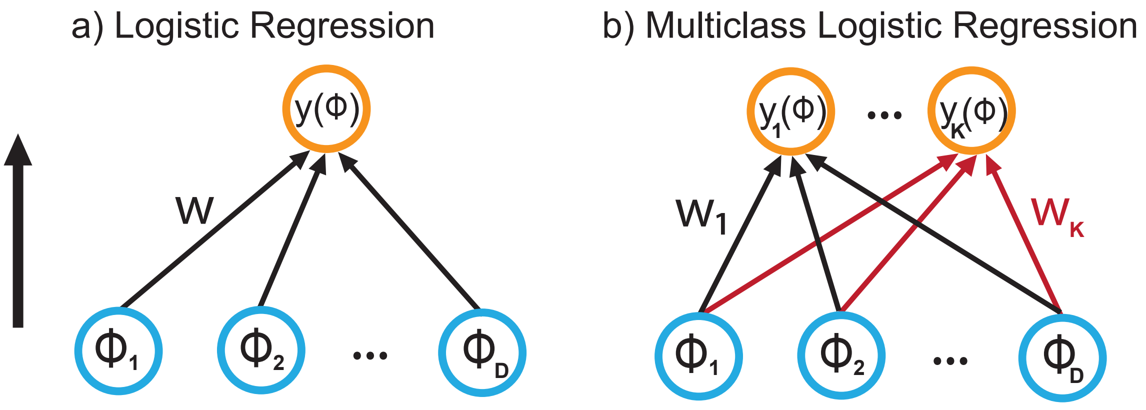

Figure 2.6: Graphical illustration of a) Logistic Regression model and b) Multiclass Logistic Regression model.

Logistic Regression is a model from statistical learning for two-class classification with as many learnable parameters (weights) as input elements, see also Figure 2.6. The input data is denoted by $X = (x_1, \dots, x_N), x_n \in \mathbb{R}^D$, it is transformed by a fixed function $\phi_n = \phi(x_n)$ beforehand, e.g. by identity, linearly or non-linearly. The transformed inputs are called features. With D weights $w = (w_1, \dots, w_D), w_i \in \mathbb{R}$, the Logistic Regression model predicts the probability of class membership to the classes $\mathcal{C}_1, \mathcal{C}_2$ as follows:

$$p(\mathcal{C}_1|\phi(x_n)) = y(\phi_n) = \sigma(w^{\mathrm{T}}\phi_n) \tag{2.11}$$

with $\sigma(a)$ being the logistic sigmoid function:

$$\sigma(a) = \frac{1}{1+\exp(-a)} \tag{2.12}$$

I.e. the dot product of feature and parameter vector is carried out - the so-called activation - and afterwards passed through the sigmoid function - a so-called activation function. By application of the logistic sigmoid function all defining attributes of a probability distribution are guaranteed. Hence $p(\mathcal{C}_2|\phi(x_n)) = 1 - p(\mathcal{C}_1|\phi(x_n))$ holds.
Logistic Regression is a supervised model. Thus, for inference we need class labels on the training dataset, which are denoted by $t_n \in \{0, 1\}$ associated with each feature vector ϕ_n, $n = 1, \dots, N$. Consider e.g. the 20 Newsgroups dataset, see Section 2.1.1. Select only entries

[8]We do not claim to sum it up any better than the Machine Learning Bible.

from two categories and use the word counts as input to the Logistic Regression. The target values need to be encoded, e.g. '0' for 'comp.graphics' and '1' for 'sci.med'. The likelihood that the model predicts correct, given the parameters, is then:

$$p(t|w) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n} \tag{2.13}$$

with $t = (t_1, \ldots, t_N)^{\mathrm{T}}$ and $y_n = p(\mathcal{C}_1|\phi_n)$. Now the question is how to determine the optimal parameter vector w. Optimization is carried out using the so-called cross-entropy error function - the negative logarithm of the likelihood function. This classic trick facilitates the optimization while not changing the position of extremes:

$$E(w) = -\ln(p(t|w)) = -\sum_{n=1}^{N}(t_n \ln y_n + (1-t_n)ln(1-y_n)) \tag{2.14}$$

with $y_n = \sigma(a_n)$ and $a_n = w^{\mathrm{T}}\phi_n$. Finally, we seek a minimum of the cross entropy error function:

$$\arg\min_{w} E(w) \tag{2.15}$$

This can be achieved iteratively by (stochastic) gradient descent. Therefore, we need the gradient of the cross entropy error function w.r.t. w:

$$\nabla E(w) = \sum_{n=1}^{N} \delta_n \phi_n \tag{2.16}$$

with $\delta_n = y_n - t_n$ constituting the model output minus the actual target values. The detailed gradient derivation can be found in Appendix C.2.

The **Multiclass Logistic Regression** extends the two-class model to several classes such that one class out of several classes $\mathcal{C}_1, \ldots \mathcal{C}_K$ is chosen to be most likely, i.e. a 1-of-K coding scheme. Therefore, the softmax function is used and the induced probability distribution becomes:

$$p(\mathcal{C}_k|\phi(X)) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{2.17}$$

with activations $a_j = w_k^{\mathrm{T}}\phi$. Recognize that we have one parameter vector w_k for each class K, i.e. K times D weights in total. The derivative of the softmax function is

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \tag{2.18}$$

with I_{kj} being the elements of the identity matrix, see Appendix C.1. Using this model the 20 Newsgroups dataset example can be extended to its full 20 categories: one record corresponds to exactly one category. The corresponding likelihood function is:

$$p(T|w_1, \dots, w_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{2.19}$$

with $y_{nk} = y_k(\phi_n)$ and T being an NxK matrix of the target variables with elements t_{nk}. Analogous to two-class Logistic Regression the cross-entropy error function for Multiclass Logistic Regression is the negative logarithm of the likelihood function:

$$E(w_1, \dots, w_K) = -\ln p(T|w_1, \dots, w_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk} \tag{2.20}$$

Again we seek a minimum to find optimal parameters w_j:

$$\arg\min_{w_j} E(w_1, \dots, w_K) \tag{2.21}$$

The gradient w.r.t. one parameter vector w_j however, yields the same result as 2.16:

$$\nabla_{w_j} E(w_1, \dots, w_K) = \sum_{n=1}^{N} \delta_{nj} \phi_n \tag{2.22}$$

with $\delta_{nj} = y_{nj} - t_{nj}$, elaborately derived in Appendix C.2. Since the gradients of Logistic and Multiclass Logistic Regression are of the form of Eq. B.3, training can be carried out using batch-, online- or minibatch gradient descent.

2.5 Feed-forward Neural Networks & error backpropagation

This Section briefly introduces Artificial Neural Networks and training them with the well known error backpropagation algorithm. In particular, we focus on fully connected Feed-forward Neural Networks arranged in homogeneous layers, with non-linear, continuous activation functions and cross-entropy or sum-of-squares-error functions corresponding to sigmoidal and softmax or identity output layers. A classical training algorithm for Artificial Neural Networks is (gradient descent) error backpropagation - it is of particular interest, since Deep Belief Networks, a special variant of Feed-forward Neural Networks, will be fine-tuned by application of this algorithm. Extending the concepts of the previous Section, this Section again closely follows Bishop [7] Chapter 5.

Artificial Neural Networks attempt to reconstruct neural processing in biological systems, e.g. in the human brain Important contributions to the development of Artificial Neural Networks were submitted by McCulloch and Pitt [34], Rosenblatt [43] and Widroff and Hoff [59]. Biological neurons collect information through input connections (dendrites), accumulate it and fire a signal through an output connection (axon) upon exceeding a certain threshold, cf. Figure 2.7 a). Moreover, axons and dendrites are connected by synapses such that multiple neurons form a network of neurons.

Nowadays, Artificial Neural Networks are a broad and widely used family of supervised statistical learning models. In contrast to their complex biological role models they are mathematical concepts and restricted in structure and time. An Artificial Neural Network consists of processing units interconnected through weights corresponding to neurons in the human brain linked by dendrites and axons. Weights (learnable parameters) along the links can be imagined as synapses. The output of a processing unit is carried out by taking the weighted linear combination of all inputs and transforming it through an activation function as illustrated in Figure 2.7 b).

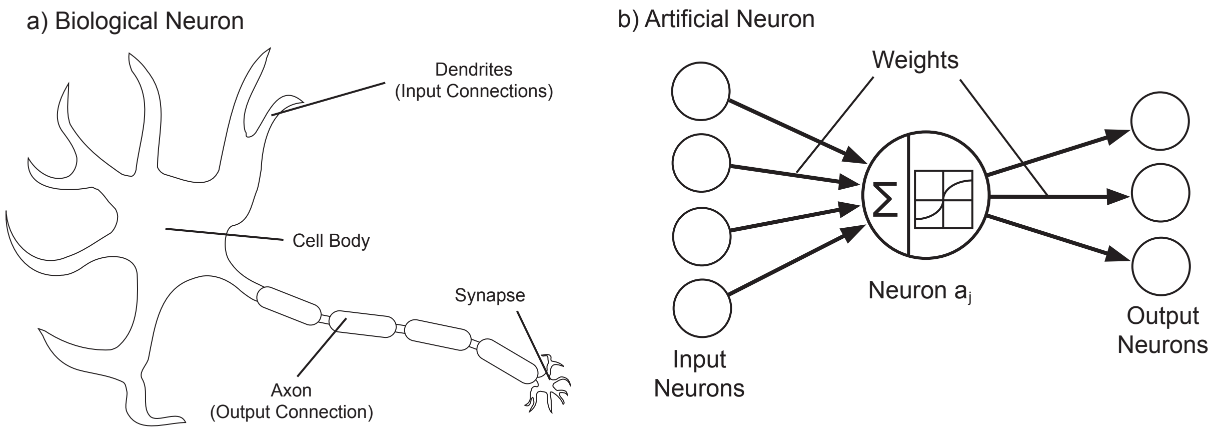

Figure 2.7: Schematic illustration of Biological and Artificial Neurons. A Biological Neuron gathers information through its input connections (dendrites), processes the information in the cell body and emits information through its output connection (axon). An Artificial Neuron comprises the same components and functionality on a mathematically formalized level: It collects information from input neurons, processes it and outputs information to successive neurons.

A single neuron for two-class classification can be mathematically formalized and simulated by Rosenblatt's famous **Perceptron** [43]. Assume a dataset X and feature vectors $\phi(X)$ given as defined in the previous Section 2.4. The Perceptron model is quite similar to the Logistic Regression model. The linear combination of feature and weight vector is calculated and passed through an activation function f:

$$y(x) = f(w^{\mathrm{T}}\phi_n(x)) \tag{2.23}$$

with f defined as follows, see also Figure 2.9:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \tag{2.24}$$

The differences are the usage of a discontinuous activation function instead of the continuous sigmoid function, using '-1' and '+1' as class labels for $\mathcal{C}_1, \mathcal{C}_2$ instead of '0' and '+1' and introducing the concept of biases. [9] Compared to the Logistic Regression the bias adds another weight w_0. Therefore, the feature vector includes a component $\phi_{n0}(x) = 1$. Hence, there are $D + 1$ learnable parameters in total. The bias weight, in geometric terms, translates the separating hyperplane of the two classes in feature space. While there exists a celebrated training algorithm for the Perceptron, we skip it, since ultimately we are interested in Deep Belief Networks and their relationships to other models.

We consider here merely so-called **Feed-forward Neural Networks**. They are also called 'Multilayer Perceptron'. However, they are a composition of Logistic Regressions rather than a composition of multiple Perceptrons. Yet, the here introduced bias concept is adopted.

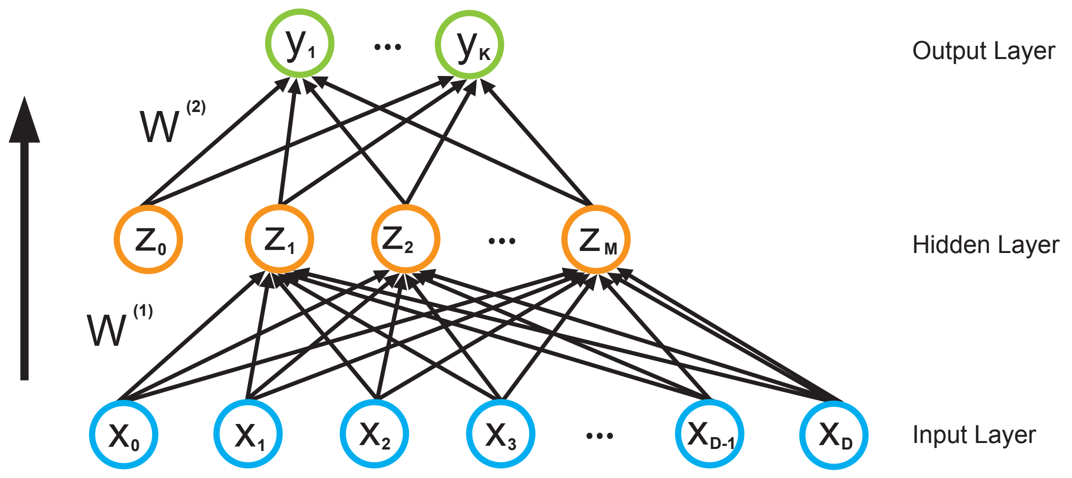

Figure 2.8: Architecture of a layered Feed-forward Neural Network with two layers of weights. Each circle represents a processing unit, i.e. an Artificial Neuron. Each connecting line represents a weight. The arrow on the left indicates the data flow through the network in forward direction.

Feed-forward Neural Networks are restricted in their structure: the data flows from input units strictly towards output units without recurrences, i.e. representable by a directed acyclic graph (DAG). Additionally, we organize the units only in homogeneous layers of neurons, i.e. equal activation functions on one layer. As depicted in Figure 2.8, the processing units are arranged in layers that are typically named input layer, hidden layer and output layer (of neurons). Note that several hidden layers of neurons can be installed. We define the Neural Network in Figure 2.8 as a two-layer network, hence the number of layers equals the number of layers of weights.

The Neural Network in Figure 2.8 predicts the output as follows. The input data X, consider e.g. again word counts extracted from the 20 Newsgroup dataset, cf. Section 2.1.1, is fed into the first layer (input layer) of neurons. The activations a_j for the hidden layer are calculated by

[9]The Logistic Regression models can be extended to employ bias parameters, too. However, it naturally fits the discussion, here.

linearly combining the input data with the first layer of weights:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{2.25}$$

The superscript (1) indicates the first layer of weights. Recognize that the bias weight $w_{j0}^{(1)}$ can either be expressed explicitly or incorporated into the sum. The value of x_0 of the input vector is clamped to 1. The output of the hidden layer neurons is computed by passing these activations through a transformation function $h^{(1)}(a)$:

$$z_j = h^{(1)}(a_j) \tag{2.26}$$

If $h^{(1)}(a_j) = \sigma(a_j)$, then this process equals the parallel execution of Logistic Regressions. The output of the hidden layer neurons serves as input for the next layer. The second-layer activations a_k are carried out likewise:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_i + w_{k0}^{(2)} = \sum_{j=0}^{D} w_{kj}^{(2)} x_i \tag{2.27}$$

with second layer bias weight $w_{k0}^{(2)}$. The model output is calculated by applying an output activation function $h^{(2)}(a_k)$ on the second layer activations:

$$y_k = a_k \tag{2.28}$$

In sum, the $y_{nk}(x_n, W)$ are calculated as follows:

$$\begin{aligned} y_{nk}(x_n, W) &= h^{(2)} \left(\sum_{m=1}^{M} w_{kj}^{(2)} h^{(1)} \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_{ni} + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) && (2.29) \\ &= h^{(2)} \left(\sum_{m=0}^{M} w_{kj}^{(2)} h^{(1)} \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_{ni} \right) \right) && (2.30) \end{aligned}$$

with $x_0 = 1$, $W = \{W^{(1)}, W^{(2)}\}$ and activation functions $h^{(1)}, h^{(2)}$.

There exists a variety of possible activation functions for $h^{(1)}$, e.g. the Perceptron activation function (also called Heaviside function) or sigmoidal functions like the logistic sigmoid function (also called Fermi function) and hyperbolic tangent. In order to apply the error backpropagation algorithm it will be wise to choose a continuously differentiable function, e.g. the hyperbolic tangent (tanh), see also Eq. C.1 in the Appendix:

$$tanh(a) := \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}; \quad \frac{\partial tanh(a)}{\partial a} = 1 - tanh^2(a) \tag{2.31}$$

Note that the sigmoid and tanh function are related: $tanh(a) = 2\sigma(a) - 1$. However, the function ranges differ: $tanh\colon \mathbb{R} \to]-1, 1[$, but $\sigma\colon \mathbb{R} \to]0, 1[$. We usually stick with the logistic sigmoid function, because it naturally fits standard Restricted Boltzmann Machines with binary ($\{0, 1\}$)

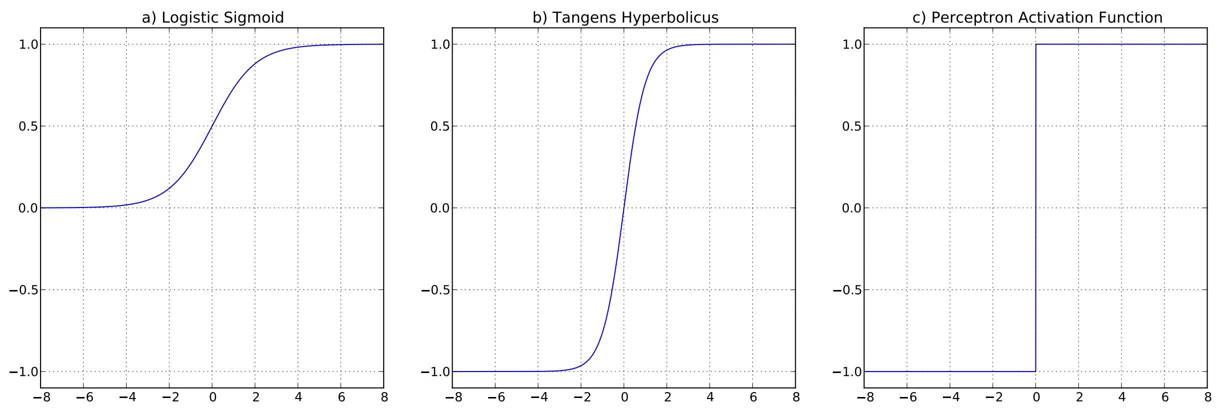

Figure 2.9: Typical activation functions for the hidden layer neurons in Artificial Neural Networks.

distributed hidden and visible units, cf. Subsection 2.7.2.

For the output layer of neurons the choice of the activation function $h^{(2)}$ depends upon the choice of the error function. Indeed, it is a general result that for a certain error function (assuming the conditional distribution of the target variables $p(t|\eta, s)$ is a member of the exponential family distributions with natural parameters η) a corresponding 'canonical' activation function exists: a so-called **canonical link function**. This is a result from Generalized linear models, see Nelder and Wedderburn [37].

Hence, for a sigmoidal output layer one uses the cross-entropy error function for Logistic Regression, for a softmax output layer the cross-entropy error function for Multiclass Logistic Regression and for the identity as output layer activation function a sum-of-squares error function. Even more interestingly, the gradient is always (up to a linear scaling factor $\frac{1}{s}$) of the form of Eq. B.3:

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{2.32}$$

hence we are permitted to apply batch-, online- or minibatch-gradient descent. For Feed-forward Neural Networks with an identity output layer, i.e. $y_{nk} = a_{nk}$, the sum-of-squares error function is utilized:

$$E_n(w) = \frac{1}{2}\|y(x_n, w) - t_n\|^2 \tag{2.33}$$

Training Feed-forward Neural Networks can be carried out using the famous **error backpropagation algorithm** introduced by Rumelhart et al. [45] in 1986. In contrast to e.g. Support Vector Machines (SVM, a 'competing' broad class of models for statistical inference) the optimization problem in Neural Networks is non-convex and consequently requires an iterative learning procedure, typically gradient descent.

The error backpropagation algorithm comprises two stages. *First*, the input layer is fed with the input data $X = (x_1, \dots, x_D), x_i \in \mathbb{R}$ and the information is propagated forward through the network along the lines connecting the processing units to calculate the model output y_{nk}, e.g. for a sigmoidal output layer using Eq. 2.29 . *Secondly*, the model error δ is propagated backwards - again along the connecting lines but in reverse direction - in order to update the weights.

Consider for the moment only a linear model with one layer of weights $y_{nk} = \sum_i w_{ki} x_{ni}$, i.e. the hidden layer of neurons in Figure 2.8 as input layer coupled with the output layer by weights $W^{(2)}$. Then the gradient for each record in the dataset and with respect to a weight w_{ki} is

$$\nabla E_n(w_{ki}) = \frac{\partial}{\partial w_{ki}} \frac{1}{2} \sum_{n=1}^{N} \|y(z_{n,w}) - t_n\|^2 = (y(z_{n,w}) - t_n) \frac{\partial}{\partial w_{ki}} y(z_{n,w}) \tag{2.34}$$

$$= (y_{nk} - t_{nk}) x_{ni} = \delta_{nk} x_{ni} \tag{2.35}$$

with

$$\delta_{nk} := y_{nk} - t_{nk} \tag{2.36}$$

being the model error, i.e. model output minus target variables.

If one now appends again a layer whose output is fed into this linear model, i.e. in total a model equivalent to Figure 2.8, the gradient with respect to the, now last layer weights $w_{ki}^{(2)}$, is unchanged. The activations carried out by the first layer then become:

$$a_{nj} = \sum_i w_{ji}^{(1)} z_i \tag{2.37}$$

and $z_{nj} = h^{(1)}(a_{nj})$. Now the question is how to update the weights in front of the last layer of weights. An answer to that is to propagate the model errors backwards through the Neural Network.

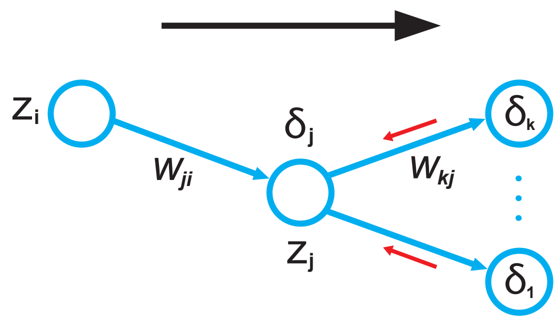

Figure 2.10: Schematic illustration of the error backpropagation for a weight $w_{ji}^{(1)}$ in front of a processing unit a_j.

As illustrated in Figure 2.10 a weight $w_{ji}^{(1)}$ is only influenced by the summed input a_{nj} to unit

j in backwards direction. Thus, the chain rule can be applied:

$$\frac{\partial E_n(w)}{\partial w_{ji}^{(1)}} = \frac{\partial E_n(w)}{\partial a_{nj}} \frac{\partial a_{nj}}{\partial w_{ji}} = \delta_{nj} z_{ni} \tag{2.38}$$

For reasons of clarity and simplification it is useful to rename both factors of the derivative: $\delta_{nj} := \frac{\partial E_n(w)}{\partial a_{nj}}$ and $z_{ni} := \frac{\partial a_{nj}}{\partial w_{ji}^{(1)}}$.

The first factor of the derivative, the δ_{nj}, comprises the 'error', i.e. derivatives of all outgoing connections that arise by application of the chain rule for partial derivatives:

$$\delta_{nj} = \frac{\partial E_n(w)}{\partial a_{nj}} = \sum_k \frac{\partial E_n}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial a_{nj}} \tag{2.39}$$

Ultimately, with the δ_{nk} given by Eq. 2.36 the backpropagation formula takes the form:

$$\delta_{nj} = \frac{\partial}{\partial a_{nj}} h^{(1)}(a_{nj}) \sum_k w_{kj}^{(2)} \delta_{nk} \tag{2.40}$$

Finally, for the example with one hidden layer and an identity function output layer of neurons, the gradients for the two weight layers amount to:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_{nj} x_{ni}, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_{nk} z_{nj} \tag{2.41}$$

The extension to multiple hidden layers is discussed in the next Section.

2.6 Deep Belief Networks (DBN) & Deep Auto-Encoders (DAE)

Deep Belief Networks (DBNs) are special Feed-forward Neural Networks. In contrast to shallow networks with only one hidden layer, a Deep Belief Network has more than one hidden layer, see Figure 2.11 b). The advantage of DBNs is that they can learn higher order correlations in a compact form. While one can train them purely with the error backpropagation algorithm, this usually ends up in poor performance. Bengio [6] e.g. suggests that with random initialization error backpropagation gets quickly stuck in poor, nearby local optima.

An effective and efficient way to train Deep Belief Networks was proposed 2006 by Hinton et al. [19], [17]. The idea is to pre-train a Deep Belief Network unsupervised with a stack of Restricted Boltzmann Machines (RBM) using Contrastive Divergence learning, sketched in Figure 2.11 a) and afterwards using the result as initialization for the error backpropagation algorithm. In this proceeding, error backpropagation constitutes fine-tuning of the Deep Belief Network.

Restricted Boltzmann Machines are discussed in detail in the next Section. We now generalize Neural Networks and the error backpropagation formula to multiple hidden layers. The weights obtain a superscript variable l in brackets numbering the weight layers, 1 being the weight layer closest to the input neurons. Then for a general weight $w_{ij}^{(l)}$ in front of a unit a_j the error

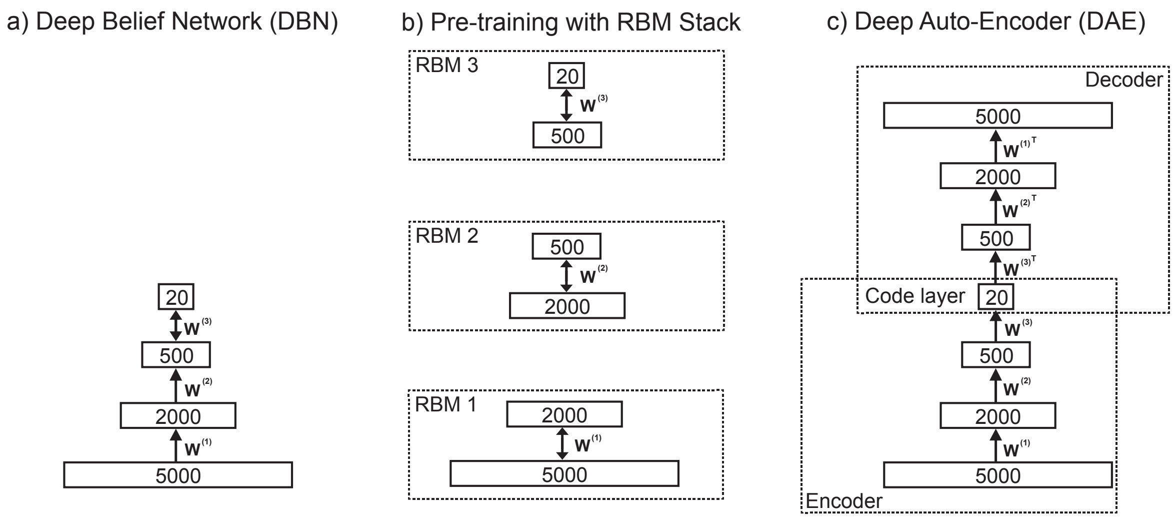
a) Deep Belief Network (DBN)
b) Pre-training with RBM Stack
c) Deep Auto-Encoder (DAE)
RBM 3
RBM 2
RBM 1
Decoder
Code layer
Encoder
20
500
2000
5000
W(1)
W(2)
W(3)
W(1)T
W(2)T
W(3)T

Figure 2.11: Schematical Illustration of a) Deep Belief Network, b) Greedy pre-training each layer of a DBN with Contrastive Divergence learning of Restricted Boltzmann Machines and c) Deep Auto-Encoders that produce small codes on the code layer when applying the input data on the input and output layers of the Deep Auto-Encoder; rectangles represent a layer of neurons, the integer within the rectangles the amount of neurons in the respective layer.

backpropagation rule becomes:

$$\delta_{nj} = \frac{\partial}{\partial a_{nj}} h^{(l)}(a_{nj}) \sum_k w_{kj}^{(l+1)} \delta_{nk} \tag{2.42}$$

together with activations z_j the derivative is:

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_{nj} z_{ni} \tag{2.43}$$

analogous to Equations 2.39 to 2.41 and Figure 2.10.

Another manifestation of Deep Belief Networks are **Deep Auto-Encoders**, presented by Hinton and Salakhutdinov [17]. The idea is to add a flipped Deep Belief Network at the output layer of a Deep Belief Network, as illustrated in Figure 2.11 c). Here, the input feature vectors are applied at both ends of the Deep Auto-Encoder. This performs a dimensionality reduction from input to code layer, since the high-dimensional input feature vectors are mapped to lower-dimensionality code vectors. It can also be thought of as a hash function. Empirical investigations show that often similar feature vectors are mapped to similar code vectors and vice versa dissimilar feature vectors map to dissimilar code vectors, cf. Salakhutdinov and Hinton [46].

2.7 Restricted Boltzmann Machines (RBM) & Harmonium models

Restricted Boltzmann Machines (RBMs), introduced by Smolensky [53], are special Markov Random Fields that form a two-layer architecture with nodes in one layer conditionally independent from nodes in the other layer. Due to their two-layer architecture they can serve as building blocks of a Deep Belief Network and thus to pre-train Deep Belief Networks unsupervised and layer-wise with an algorithm called Contrastive Divergence (CD) presented by Hinton [18]. Several stand-alone RBM models have been derived, also for information retrieval.
This Section first introduces the standard Restricted Boltzmann Machine, also called Harmonium, using Bernoulli units. It has been extended by Welling et al. to support distributions from the Exponential Family, so called Exponential Family Harmonium (EFH)) [58]. From this point on, a chain of several distribution combinations has been explored with application in the field of document retrieval: Undirected Probabilistic Latent Semantic Indexing by Hofmann [58], Rate Adapting Poisson by Gehler et al. [15], Constrained Poisson [46] and Replicated Softmax (RSM) [47] both presented by Salakhutdinov and Hinton. In particular, we put the spotlight on the Replicated Softmax (RSM) model, which is to be used in our Neural Networks. Furthermore, this Section describes how the EFH model has been extended to support multiple different exponential family member distributions on the visible layer, so-called Dual Wing Harmoniums (DWH) by Xing et al. [62]. Last but not least, parts of Hinton's practical guidelines for Contrastive Divergence training are recapitulated [16].

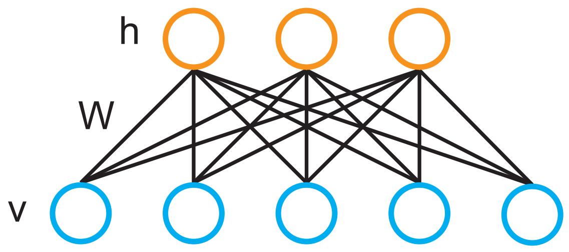

Figure 2.12: The general structure of a Restricted Boltzmann Machine can be represented by a (fully) bipartite, undirected graph.

In 1986, Smolensky [53] presented harmony theory that introduces **Restricted Boltzmann Machines (Harmoniums)**. An undirected graph, with nodes representing random variables and the links representing probabilistic relationships among the random variables, is called a Markov Random Field (MRF), [10], see for example Bishop [7] Section 8.3. A RBM then can be described by an *undirected* graph consisting of nodes organized in two layers: one layer of so-called *hidden* random variables and one layer of so-called *visible* random variables that are coupled by weights along the connecting lines, see Figure 2.12. The nodes are also called units. Due to this (fully) bipartite structure, Restricted Boltzmann Machines are an undirected counterpart to slices of a Deep Belief Network.

[10]They are called Markov Random Fields, as the undirected connections reflect that the Markov property, that is conditional independence of one random variable given the others, is fulfilled.

Moreover, Restricted Boltzmann Machines are **energy-based models**, i.e. the joint probability distribution over the random variables is of the form:

$$p(v,h) = \frac{1}{Z}\exp(-E(v,h)) = \frac{\exp(-E(v,h))}{\sum_{\tilde{v},\tilde{h}}\exp(-E(\tilde{v},\tilde{h}))} \tag{2.44}$$

with $E(h,v)$ being a so-called energy function. The so-called partition function $Z := \sum_{\tilde{v},\tilde{h}}\exp(-E(\tilde{v},\tilde{h}))$ normalizes the exponential of the negative energies to a probability distribution by summing (or integrating for continuous random variables) over all possible configurations, i.e. all possible values the random values can be assigned to. The energies are said to 'work in the log-probability domain', cf. Bengio [5] Section 5.1.

Using basic probability theory laws, the probability $p(v)$ can be obtained by marginalizing out h:

$$p(v) = \sum_{\hat{h}} p(v,\hat{h}) = \sum_{\hat{h}} \frac{\exp(-E(v,\hat{h}))}{\sum_{\tilde{v}}\sum_{\tilde{h}}\exp(-E(\tilde{v},\tilde{h}))} \tag{2.45}$$

The ansatz for the conditional probability distribution $p(h|v)$ becomes:

$$p(h|v) = \frac{p(v,h)}{p(v)} = \frac{\frac{\exp(-E(v,h))}{\sum_{\tilde{v}}\sum_{\tilde{h}}\exp(-E(\tilde{v},\tilde{h}))}}{\sum_{\hat{h}}\frac{\exp(-E(v,\hat{h}))}{\sum_{\tilde{v}}\sum_{\tilde{h}}\exp(-E(\tilde{v},\tilde{h}))}} = \frac{\exp(-E(v,h))}{\sum_{\hat{h}}\exp(-E(v,\hat{h}))} \tag{2.46}$$

Equation 2.44 is a Boltzmann distribution, hence the name Restricted Boltzmann Machine. The Boltzmann distribution stems from physics, where it describes the states of dynamical systems. At least from an imagination point of view, low energies are desirable, e.g. water forms balls in air since this shape has the lowest surface tension. Each possible configuration of the random variables is assigned a scalar probability.[11] Due to the negative sign in the exponential, states with low energy are assigned a high probability. Learning, in a machine learning sense, then equals reshaping the probability distribution into desirable forms, see Bengio [5]. In the case of RBMs this means reconstructing the input data as good as possible.
In contrast to a General Boltzmann Machine the energy function of a standard Restricted Boltzmann Machine takes the form of a first-order polynomial:

$$E(v,h,) = -v^{\mathrm{T}}Wh - a^{\mathrm{T}}v - b^{\mathrm{T}}h \tag{2.47}$$

where v, h correspond to the random variables in the nodes of the undirected graph, W is a coupling matrix and a, b are bias variables. Each random variable has one bias variable attached to it. The restriction compared to a General Boltzmann Machine can be thought of as to forbid any links connecting hidden with hidden variables or visible/visible variable links. Again, Restricted Boltzmann Machines correspond to (fully) bipartite graphs.

[11] for discrete random variables

2.7.1 Product of Experts (PoE) & Contrastive Divergence (CD)

In 2002 Hinton [18] presented an algorithm called Contrastive Divergence (CD) to train **Product of Expert (PoE)** models. Since a standard Restricted Boltzmann Machine is a Product of Experts model, to be shown later on, it can be trained with Contrastive Divergence. We use Wood's excellent derivation of CD. [61]

A PoE model combines individual experts by multiplication rather than by summation. Models that use summation are called mixture models, e.g. Mixture of Gaussians. They are of the form:

$$p(X|\theta_1,\dots,\theta_E) = \sum_{e=1}^{E} \pi_e p(X|\theta_e) \tag{2.48}$$

where the π_e are mixing coefficients that ensure normalization to a probability distribution, i.e. $\sum_{e=1}^{E} \pi_e = 1$ must be satisfied. In contrast to that a PoE model, with E individual experts, expresses the probability of the input data X with N records as follows:

$$p(X|\theta_1,\dots,\theta_E) = \prod_{n=1}^{N} \frac{\prod_{e=1}^{E} p_e(x_n|\theta_e)}{\sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)} \tag{2.49}$$

where the denominator contains a summation of all possible configurations $\tilde{x}$ of the expert's random variables, which ensures normalization to a probability distribution. According to Hinton, the advantage of a factorial distribution is that it is sharper. If one expert turns to zero, then the overall probability becomes zero, thus the experts can focus on single constraints. The classical idea for learning the model parameters would be the application of maximum likelihood learning. Therefore, one needs to calculate the gradient w.r.t. the parameters of one expert θ_m of the log likelihood:

$$\frac{\partial}{\partial \theta_m} \log p(X|\theta_1,\dots,\theta_E) = \frac{\partial}{\partial \theta_m} \log \prod_{n=1}^{N} \frac{\prod_{e=1}^{E} p_e(x_n|\theta_e)}{\sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)} \tag{2.50}$$

Using $log(a * b) = log(a) + log(b)$ it follows that:

$$\frac{\partial \log \prod_{n=1}^{N} p(x_n|\theta_1,\dots,\theta_E)}{\partial \theta_m} = \frac{\partial \log \prod_{n=1}^{N} \prod_{e=1}^{E} p_e(x_n|\theta_e)}{\partial \theta_m} - \frac{\partial \log \prod_{n=1}^{N} \sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)}{\partial \theta_m} \tag{2.51}$$

$$\sum_{n=1}^{N} \frac{\partial \log p(x_n|\theta_1,\dots,\theta_E)}{\partial \theta_m} = \sum_{n=1}^{N} \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} - N \frac{\partial \log \sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)}{\partial \theta_m} \tag{2.52}$$

The last term of Eq. 2.52 can be reformulated as follows:

$$N \frac{\partial \log \sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)}{\partial \theta_m} \tag{2.53}$$

$$= N \frac{1}{\sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)} \frac{\partial \sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)}{\partial \theta_m} \tag{2.54}$$

$$= N \frac{1}{\sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)} \sum_{\tilde{x}} \prod_{e=1, e\neq k}^{E} p_e(\tilde{x}|\theta_e) \frac{\partial p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.55}$$

$$= N \frac{1}{\sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e)} \sum_{\tilde{x}} \prod_{e=1}^{E} p_e(\tilde{x}|\theta_e) \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.56}$$

$$= N \sum_{\tilde{x}} p(\tilde{x}|\theta_1, \ldots, \theta_E) \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.57}$$

$$= N \sum_{\tilde{x}} p(\tilde{x}|\theta_1, \ldots, \theta_E) \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.58}$$

using $\frac{\partial p}{\partial x} = p \frac{\partial log\, p}{\partial x}$. Inserted into Eq. 2.52 the log likelihood amounts to:

$$\sum_{n=1}^{N} \frac{\partial \log p(x_n|\theta_1, \ldots, \theta_E)}{\partial \theta_m} = \sum_{n=1}^{N} \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} - N \sum_{\tilde{x}} p(\tilde{x}|\theta_1, \ldots, \theta_E) \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.59}$$

Unfortunately, exact inference is not tractable due to the exponentially many possible configurations of the partition function, the last term of Eq. 2.59. It can, though, be approximated using Gibbs sampling. Gibbs sampling, a Markov Chain Monte Carlo method, see e.g. Bishop [7] Section 11.2. to 11.3., or Resnik and Hardisty [42], is an algorithm to sample from complex distributions. For the conditionally independent random variables it takes a particular simple form. Nevertheless, running the Gibbs chain to an equilibrium state is still computationally painful.

Here comes **Contrastive Divergence** into action. Assuming the input records to be i.i.d., Eq. 2.59 can be expressed solely in terms of expectations:

$$N \sum_{n=1}^{N} \frac{1}{N} \frac{\partial \log p(x_n|\theta_1, \ldots, \theta_E)}{\partial \theta_m} = N \sum_{n=1}^{N} \frac{1}{N} \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} - N \sum_{\tilde{x}} p(\tilde{x}|\theta_1, \ldots, \theta_E) \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \tag{2.60}$$

$$N \sum_{n=1}^{N} p^0(x_n) \frac{\partial \log p(x_n|\theta_1, \ldots, \theta_E)}{\partial \theta_m} = N \sum_{n=1}^{N} p^0(x_n) \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} - N \langle \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \rangle_{p^\infty} \tag{2.61}$$

$$\left\langle \frac{\partial \log p_\theta^\infty}{\partial \theta_m} \right\rangle_{p^0} = \left\langle \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} \right\rangle_{p^0} - \left\langle \frac{\partial \log p_e(\tilde{x}|\theta_m)}{\partial \theta_m} \right\rangle_{p_\theta^\infty} \tag{2.62}$$

where $\langle f \rangle_p := \sum_x p(x)f(x)$ denotes an expectation of a function $f(x)$ under a probability distribution $p(x)$. Additionally, $p(x_n|\theta_1,\ldots,\theta_E)$ is renamed as p_x^∞. Now, the first term on the right hand side of Eq. 2.62 is an expectation with respect to the data distribution, whereas the second term represents the model's expectation. Since learning can be understood as reshaping the PoE model's parameters such that they model the data in 'the best possible' way, the expectation of the model shall approximate the expectation of the data distribution.
The log likelihood is intractable. Consequently, one seeks for another objective function. The Kullback-Leibler divergence (KL) is a similarity-measure for two probability distributions $p(x), q(x)$ (see e.g. Bishop [7] Subsection 1.6.1.):

$$p\Big|\Big|q := -\sum_x p(x)\log\frac{q(x)}{p(x)} \tag{2.63}$$

An equivalent objective function to log likelihood is the KL divergence between p^0 and p_θ^∞:

$$p^0\Big|\Big|p_\theta^\infty = \sum_x \log\frac{p^0(x)}{p_\theta^\infty(x)} = \sum_x p^0(x)\log p^0(x) - \sum_x p^0(x)\log p_\theta^\infty(x) \tag{2.64}$$

$$= H(p^0) - \left\langle \frac{\partial \log p_\theta^\infty(x)}{\partial\theta_m} \right\rangle_{p^0} \tag{2.65}$$

where $H(x) = -\sum_x p(x)\log p(x)$ is the entropy and does not depend on the model parameters θ. Looking at the derivatives w.r.t. θ_m:

$$\frac{\partial p^0\Big|\Big|p_\theta^\infty}{\partial\theta_m} = -\left\langle \frac{\partial \log p_\theta^\infty(x)}{\partial\theta_m} \right\rangle_{p^0} \tag{2.66}$$

it follows that maximizing log likelihood and minimizing the Contrastive Divergence is equivalent. Nevertheless, running the Gibbs chain to equilibrium for approximating p_θ^∞ is still computationally awful. Therefore, Contrastive Divergence minimizes yet another objective function where p_θ^∞ cancels out:

$$\frac{\partial}{\partial\theta_m}\left(p^0\Big|\Big|p_\theta^\infty - p_\theta^k\Big|\Big|p_\theta^\infty\right) \propto \left\langle \frac{\partial \log p_m(x_n|\theta_m)}{\partial\theta_m} \right\rangle_{p^0} - \left\langle \frac{\partial \log p_m(x_n|\theta_m)}{\partial\theta_m} \right\rangle_{p_\theta^k} \tag{2.67}$$

This process can also be thought of running the Gibbs chain only for k steps and this is indeed computationally tractable and this is called k-step Contrastive Divergence, or just CD-k. As Hinton showed empirically, few, often only 1 full Gibbs steps are sufficient. Note that for equivalence, the chain rule for the k Gibbs steps needs to be taken into account, e.g. for CD-1 Eq. 2.67 misses the term $\frac{\partial p^1||p_\theta^\infty}{\partial p^1}\frac{\partial p^1}{\partial\theta_m}$ such that equality rather than proportionality is fulfilled. However, Hinton asserts that this term is usually neglectable small. Also, the Contrastive Divergence can never turn negative, since p^1 is closer to the equilibrium state than p^0. Additionally, he argues, by running the Gibbs chain only for a few steps, it is less likely to swing off the real data distribution, i.e. the parameter updates have lower variance. In summary, the k-step CD algorithm involves two major ideas: start the Gibbs chain at the input data and run it only for a few steps in order to calculate the parameter updates:

$$\delta\theta_m \propto \epsilon_t \left(\left\langle \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} \right\rangle_{p^0} - \left\langle \frac{\partial \log p_m(x_n|\theta_m)}{\partial \theta_m} \right\rangle_{p^k} \right) \tag{2.68}$$

with a learning rate ϵ_t chosen by the user, see also Figure 2.13.

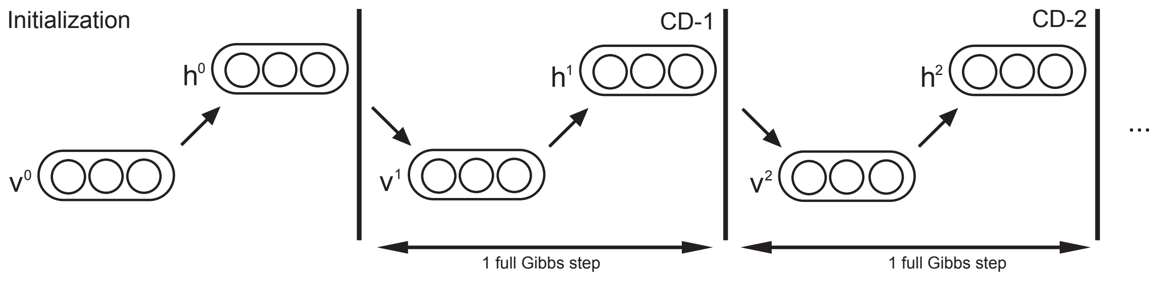

Figure 2.13: Contrastive Divergence starts the Gibbs chain at the data and runs it only for a few steps. A full Gibbs step is to sample from both layers successively.

Now, if for RBM models in the following Subsections the hidden variables are conditionally independent given the visible variables and vice versa, CD learning can be applied. Hence, a major point of interest is to show conditional independence for the following models. Last but not least, as Hinton [18] proposed, the k-step CD algorithm can be implemented using minibatches.

2.7.2 Standard RBM

In 1986, Smolensky [53] modeled a cognitive system. He called the hidden variables 'knowledge atoms' and the visible variables 'representational features'. However, the representational features could assume the values '1' for present, '0' for unspecified and '-1' for absent. In 1994, Freund and Haussler [14] introduced the so-called *influence combination machine*. They adopted Smolensky's RBM, but omitted the unspecified state. Thus, with '-1' and '+1' for visible units and '0' and '+1' for hidden units only., i.e. using Bernoulli distributions. Moreover, they showed that in this setting, hidden variables factorize conditionally independent given the visible variables and vice versa. In 2002 Hinton introduced Contrastive Divergence learning for Product of Expert [18] models and applied it to train Restricted Boltzmann Machines with Bernoulli random variables on both layers - however using the values '0' and '1', to which we will stick, too. See Figure 2.14 for a graphical illustration.

The energy function of a standard Restricted Boltzmann Machine then is of the form:

$$E(v,h) = -\sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - \sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{M} b_i h_i \tag{2.69}$$

$$= -v^{\mathrm{T}} W h - a^{\mathrm{T}} v - b^{\mathrm{T}} h \tag{2.70}$$

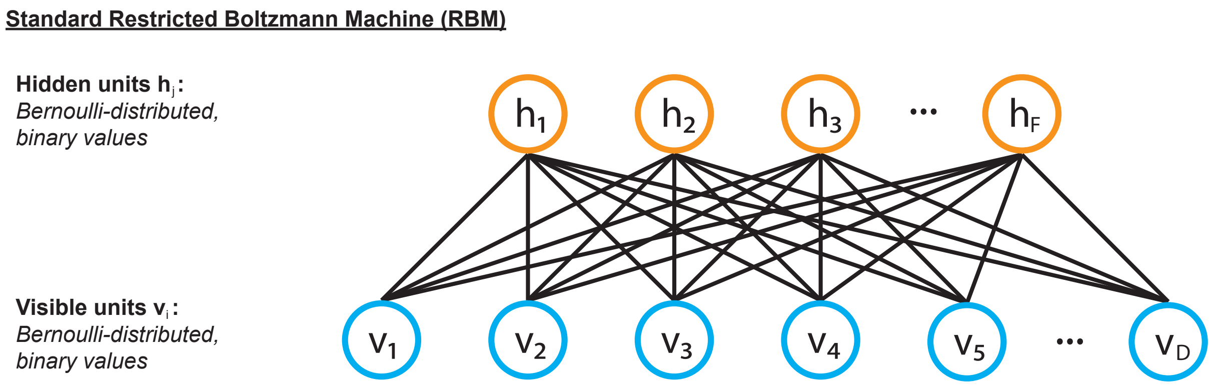
Standard Restricted Boltzmann Machine (RBM)
Hidden units h_j: Bernoulli-distributed, binary values
Visible units v_i: Bernoulli-distributed, binary values
h_1
h_2
h_3
...
h_F
v_1
v_2
v_3
v_4
v_5
...
v_D

Figure 2.14: Architecture of a standard Restricted Boltzmann Machine.

with visible units vector $v = (v_1, \ldots, v_D), v_i \in \{0,1\}$, hidden units vector $h = (h_1, \ldots, h_F), h_j \in \{0,1\}$, bias vector for the visible units $a = (a_1, \ldots, a_D), a_i \in \mathbb{R}$, bias vector for the hidden units $b = (b_1, \ldots, b_F), b_i \in \mathbb{R}$ and $W_{ij} \in \mathbb{R}$ being the coupling weight matrix.
As shown in appendix C.3, when using this energy function, the hidden variables factorize indeed conditionally independent, given the visible variables, and obey (conditional) Bernoulli distributions:

$$p(v|h) = \prod_{i=1}^{N} p(v_i|h) = \prod_{i=1}^{N} Bern\left(\sigma\left(\sum_{j=1}^{M} W_{ij} h_j + a_i\right)\right) \tag{2.71}$$

Vice versa:

$$p(h|v) = \prod_{j=1}^{M} p(h_j|v) = \prod_{j=1}^{M} Bern\left(\sigma\left(\sum_{i=1}^{N} W_{ij} v_i + b_j\right)\right) \tag{2.72}$$

Additionally, sampling from the random variables is as easy as to calculate the conditional probabilities and to perform 'less than' operations:

$$p(v_i = 1|h) = \sigma\left(\sum_{j=1}^{M} W_{ij} h_j + a_i\right) \; ; \quad p(h_j = 1|v) = \sigma\left(\sum_{i=1}^{N} W_{ij} v_i + b_j\right) \tag{2.73}$$

In order to learn the weights one feeds the standard RBM model's visible vectors with the input data x_n and performs CD-k learning. From inserting $\frac{\partial E(v,h)}{\partial w_{ij}} = -vih_j$, $\frac{\partial E(v,h)}{\partial W} = -vh^{\mathrm{T}}$ respectively, into Eq. 2.68 the learning rule amounts to:

$$\delta W \propto \epsilon_t \left(\left\langle \frac{\partial \log p_m(x_n|W)}{\partial W} \right\rangle_{p^0} - \left\langle \frac{\partial \log p_m(x_n|W)}{\partial W} \right\rangle_{p^k} \right) \tag{2.74}$$

$$= \epsilon_t \left(\left\langle vh^{\mathrm{T}} \right\rangle_{p^0} - \left\langle vh^{\mathrm{T}} \right\rangle_{p^k} \right) \tag{2.75}$$

For the biases, the update rules amount to:

$$\delta a \propto \epsilon_t \left(\langle v \rangle_{p^0} - \langle v \rangle_{p^k} \right); \quad \delta b \propto \epsilon_t \left(\langle h \rangle_{p^0} - \langle h \rangle_{p^k} \right) \tag{2.76}$$

Since the model is symmetric, it is also very easy to use more than one full Gibbs step, explored by Salakhutdinov and Murray [48].

2.7.3 Exponential Family Harmonium (EFH)

In 2005, Welling et al. extended the standard RBM to use any member of the exponential family for the hidden and visible random variables, the so-called Exponential Family Harmonium [58]. See Figure 2.15 for an overview.

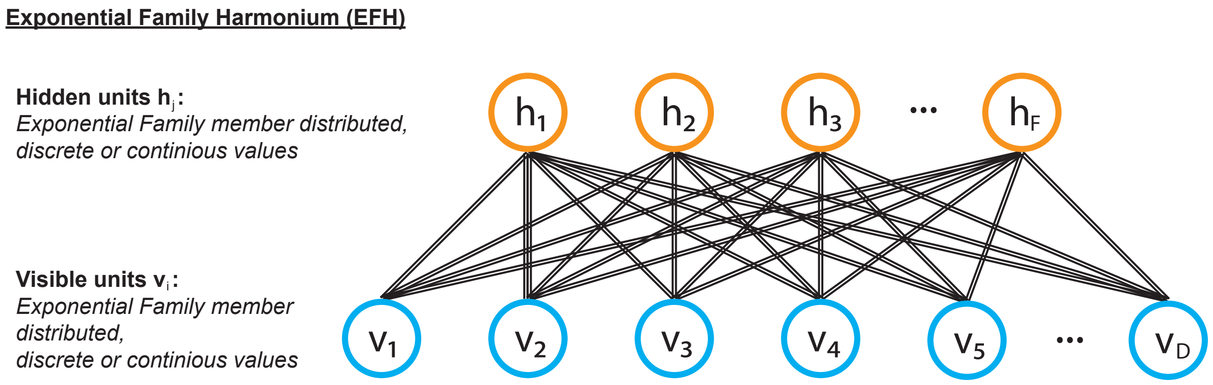
Exponential Family Harmonium (EFH)
Hidden units h_j:
Exponential Family member distributed, discrete or continious values
Visible units v_i:
Exponential Family member distributed, discrete or continious values
h1
h2
h3
...
hF
v1
v2
v3
v4
v5
...
vD

Figure 2.15: Architecture of a general Exponential Family Harmonium.

To appreciate this, assume one member of the exponential family for the visible variables and combine D many multiplicatively:

$$p(\{v_i\}) = \prod_{i=1}^{D} r_i(v_i) \exp \left(\sum_a \theta_{ia} f_{ia}(v_i) - A_i(\{\theta_{ia}\}) \right) \tag{2.77}$$

with A_i the log-partition function, $f_{ia}(v_i)$ the sufficient statistics and θ_{ia} the natural parameters, see e.g. Bishop [7] Section 2.4. and vice versa one member of the exponential family for the hidden variables and combine F many multiplicatively:

$$p(\{h_j\}) = \prod_{j=1}^{F} s_j(h_j) \exp \left(\sum_b \lambda_{jb} g_{jb}(h_j) - B_j(\{\lambda_{jb}\}) \right) \tag{2.78}$$

with B_j the log-partition function, $g_{jb}(h_j)$ the sufficient statistics and λ_{jb} the natural parameters. If these are combined consistently in an energy function with a coupling matrix W_{ia}^{jb} as follows:

$$E(v,h) = -\sum_{ia} \theta_{ia} f_{ia}(v_i) - \sum_{jb} \lambda_{jb} g_{jb}(h_j) - \sum_{ijab} W_{ia}^{jb} f_{ia}(v_i) g_{jb}(h_j) \tag{2.79}$$

then, the hidden variables given the visible factorize conditionally independent:

$$p(h|v) = \prod_{i=1}^{D} \exp\left(\sum_a \hat{\theta}_{ia} f_{ia}(v_i) - A(\{\hat{\theta}_{ia}\})\right) \tag{2.80}$$

which likewise holds for the visible given the hidden variables. See Appendix C.4 for the derivation. Note that it might be necessary to restrict W_{ia}^{jb}, e.g. $W_{ia}^{jb} \geq 0$ for some combinations of exponential family distributions in order to guarantee $p(v,h)$ is normalizable. This opens up for a wide variety of different derived EFH models and again they can be trained using (minibatch) k-step CD-k. Members of the exponential family are among others the Bernoulli, Binomial, Multinomial, Gaussian, Poisson, Exponential, Dirichlet, Beta and Weibull distributions.

2.7.4 Undirected Probabilistic Latent Semantic Indexing (UP-LSI)

Welling et al. also proposed a first application of the EFH model for document retrieval - a derived standalone model called UP-LSI.[12] They model an undirected counterpart to the well known directed information retrieval models Probabilistic Latent Semantic Indexing (pLSI) by Hofmann [20] and Latent Dirichlet Allocation (LDA) by Blei et al. [8]. See Figure 2.16 for a quick overview on the UP-LSI model.

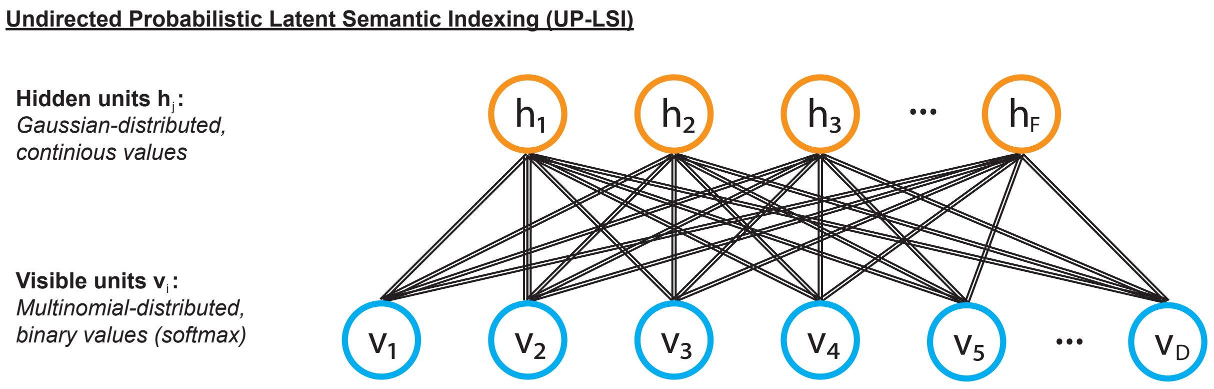
Undirected Probabilistic Latent Semantic Indexing (UP-LSI)
Hidden units h_j: Gaussian-distributed, continious values
Visible units v_i: Multinomial-distributed, binary values (softmax)
h_1
h_2
h_3
...
h_F
v_1
v_2
v_3
v_4
v_5
...
v_D

Figure 2.16: Architecture of the Undirected Probabilistic Latent Semantic Indexing Topic for topic retrieval.

The hidden units are chosen to be unit-variance Gaussians in order to model continuous latent topics:

$$p(h|v) = \prod_{j=1}^{F} \mathcal{N}\left(\sum_{ia} W_{ia}^{j} v_{ia}, 1\right) \tag{2.81}$$

Multinomial distributions, using the softmax function (1-of-K coding scheme) for the visible variables, model the input data, which are discrete word-counts (v_{ia} indicates keyword i was

[12]the name UP-LSI stems from Gehler et al. [15]

observed a times):

$$p(v|h) = \prod_{i=1}^{D} \mathcal{S}\left(\alpha_{ia} + \sum_j W_{ia}^j h_j\right) \tag{2.82}$$

where α is the bias vector for the visible units. The authors point out another interesting aspect, namely that their model is a generalization of factor analysis into undirected models on the discrete domain. Indeed, Marks and Movellan [33] showed that a diffusion network, i.e. an EFH with Gaussian visible and hidden units is equivalent to a factor analysis.

2.7.5 Rate Adapting Poisson (RAP)

Gehler et al. [15] picked up on the idea of EFH and UP-LSI and introduced the Rate Adapting Poisson model for information retrieval. In contrast to UP-LSI, they model the word counts with conditional Poisson distributions for the visible units. Moreover, the hidden units representing latent topics, are modeled with conditional Binomial distributions. See Figure 2.17 for a short overview of the model.

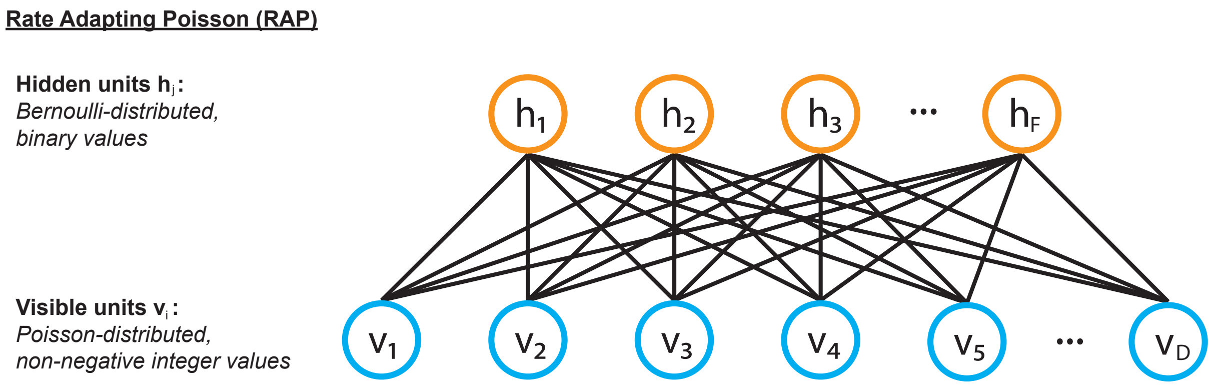
Rate Adapting Poisson (RAP)
Hidden units h_j: Bernoulli-distributed, binary values
Visible units v_i: Poisson-distributed, non-negative integer values
h1
h2
h3
...
hF
v1
v2
v3
v4
v5
...
vD

Figure 2.17: Architecture of the Rate Adapting Poisson model for information retrieval (and object recognition).

Here, the energy function is given by:

$$E(v,h) = -\sum_i [\log(\lambda_i) v_i + \log(v_i!)] \tag{2.83}$$

$$-\sum_j \left[\log\left(\frac{p_j}{1-p_j}\right) + \log(h_j!) + \log((M_j - h_j)!)\right] - \sum_{i,j} v_i h_j W_{ij} \tag{2.84}$$

with M_j being the total number of samples and p_j the probability of success for topic j. λ_i denotes the mean rate for the conditional Poisson distribution of word i. Given the energy function the visible and hidden conditional distributions factorize to:

$$p(v|h) = \prod_{i=1}^{D} Pois_{v_i}\left(\exp\left(\log(\lambda_i) + \sum_{j=1}^{F} W_{ij} h_j\right)\right) \tag{2.85}$$

[13] and:

$$p(h|v) = \prod_{j=1}^{F} Bin_{h_j} \left(\sigma \left(\log \left(\frac{p_j}{1-p_j} \right) + \sum_{i=1}^{D} W_{ij} v_i \right), M_j \right) \tag{2.86}$$

Due to the shifting of the parameters performed in Equations 2.85 and 2.86, the model is named rate adapting. The RAP model reduces the amount of parameters in comparison to the UP-LSI model. This is due to using single weight matrix W_{ij} rather than 'multiple matrices' W_{ia}^{j} with j indexing matrices and by using Poisson rates instead of a Multinomial distributions.

2.7.6 Constrained Poisson Model (CP) & Semantic Hashing (SH)

Salakhutdinov and Hinton [46] carried on the idea of the Rate Adapting Poisson model and introduced the Constrained Poisson model for information retrieval. First, the hidden variables are Bernoulli distributed units, i.e. a latent topic is either on or off. Secondly, the visible units are still Poisson distributed, but for reasons of numerical stability the Poisson rates, i.e. visible variables given the hidden variables are multiplied by the factor $\frac{N}{\sum_k \exp(\lambda_k + \sum_j h_j W_{kj})}$ where $N = \sum_i v_i$ is the total number of words in the document (i.e. the document length). See Figure 2.18 for a graphical overview of the model. The corresponding energy function is given by:

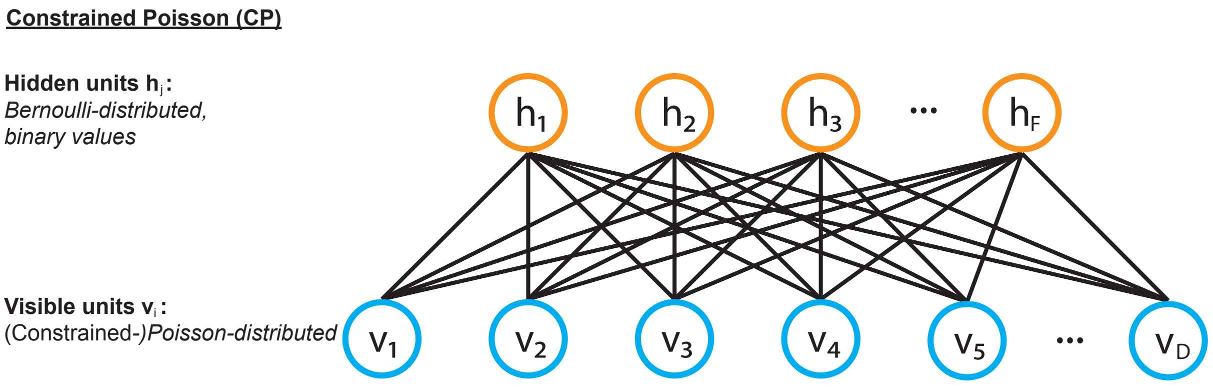

Figure 2.18: Architecture of the Constrained Poisson model.

$$E(v,h) = -\sum_i \lambda_i v_i - \sum_i \log(v_i!) - \sum_j b_j h_j - \sum_{i,j} v_i h_j W_{ij} \tag{2.87}$$

where the λ_i represent the bias of the conditional Poisson distribution for word i and b_j is the bias for topic j. Consequently, the conditional distributions factorize as follows:

$$p(h|v) = Bern\sigma \left(\left(b_j + \sum_i W_{ij} v_i \right) \right) \tag{2.88}$$

[13]This Equation is correct!

and

$$p(v_i = n|h) = Pois\left(n, \frac{\exp(\lambda_i + \sum_j h_j W_{ij})}{\sum_k \exp(\lambda_k + \sum_j h_j W_{kj})} N\right) \tag{2.89}$$

with $Pois(n, \lambda) = e^{-\lambda}\lambda^n/n!$ and the factor $\frac{N}{\sum_k \exp(\lambda_k + \sum_j h_j W_{kj}}$ is multiplicatively added. The denominator improves upon numerical stability. The Poisson rates are scaled by N to document length, $\sum_i \lambda_i = N$. This is an important aspect, since it makes the model capable of dealing with documents of different lengths.

Additionally, Salakhutdinov introduced in the same paper the idea of **Semantic Hashing**, which is put shortly to train a Deep Auto-Encoder (layers with 2000-500-500-128/20 hidden units) with a Constrained Poisson model input layer. As described in Section 2.6 this performs a dimensionality reduction from the word counts of documents as input to codes of small size (selectable by the user). The authors also empirically showed that this method acts like a hashing, i.e. similar documents end up in similar codes and dissimilar ones do not. Due to this it is called Semantic Hashing. Document retrieval can be efficiently carried out using e.g. Hamming distances between the binary codes.

2.7.7 Replicated Softmax (RSM)

The Replicated Softmax (RSM) model has been presented by Salakhutdinov and Hinton [47], too. This model for information retrieval takes word counts as input and models them with one Multinomial visible unit together with binary latent topic hidden units. A great advantage of this model is the capability to deal with documents of different lengths by scaling the hidden variables biases.

To start, consider the following energy function for one document with D keywords represented by a binary dictionary matrix $V_i^k \in \{0, 1\}$, indicating word i is keyword k together with coupling weight tensor W_{ij}^k and F hidden units:

$$E(V, h) = -\sum_{i=1}^{D}\sum_{j=1}^{F}\sum_{k=1}^{K} h_j W_{ij}^k V_i^k - \sum_{i=1}^{D}\sum_{k=1}^{K} V_i^k a_i^k - \sum_{j=1}^{F} h_j b_j \tag{2.90}$$

$$= -\sum_{k=1}^{K} h^{\mathrm{T}} W^k V^k - \left(a^k\right)^{\mathrm{T}} V^k - h^{\mathrm{T}} b \tag{2.91}$$

For this energy function, it holds that the conditional distribution for the visible variables given the hidden units factorizes and sampling is executed using:

$$p(V_i^k = 1|h) = \frac{\exp(a_i^k + \sum_{j=1}^{F} h_j W_{ij}^k)}{\sum_{q=1}^{K} \exp(a_i^q + \sum_{j=1}^{F} h_j W_{ij}^q)} \tag{2.92}$$

which is the softmax function with $a_i^k + \sum_{j=1}^{F} h_j W_{ij}^k$ as argument. It defines the appropriate 1-of-K coding scheme. See Appendix C.5 for the derivation. The hidden units model binary latent topics, i.e. are conditional Bernoulli distributed and sampling is carried out as follows:

$$p(h_j = 1|V) = \sigma\left(b_j + \sum_{i=1}^{D}\sum_{k=1}^{K} W_{ij}^k V_i^k\right) \tag{2.93}$$

Now, if one clamps the weights for all i to be equal, i.e. the weight tensor is reduced to a weight matrix - the authors call this **weight sharing**, then the model can be significantly simplified. Since the weights are equal, we can sum up over V_i^k horizontally and the binary keyword indicator variables end up in word counts, see Figure 2.19 for an illustration. In other words, by using weight sharing a new Restricted Boltzmann Machine arises with only one visible unit with a Multinomial distribution that is sampled D times, hence takes word counts as input rather than binary keyword indicator variables.

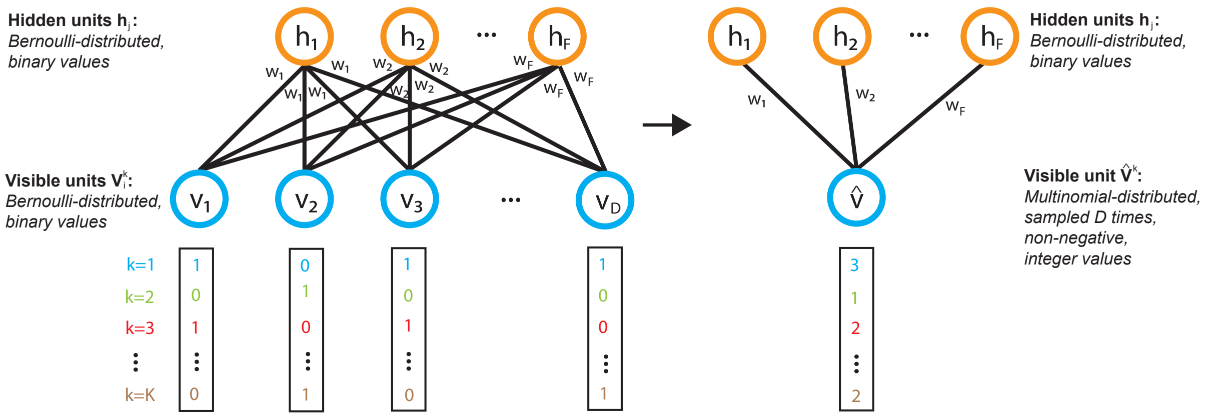
Hidden units h_j:
Bernoulli-distributed, binary values
h_1
h_2
h_F
w_1
w_2
w_F
Visible units V_i^k:
Bernoulli-distributed, binary values
v_1
v_2
v_3
v_D
k=1
k=2
k=3
k=K
1 0 1 0
0 1 0 1
1 0 1 0
1 0 0 1
h_1
h_2
h_F
Hidden units h_j:
Bernoulli-distributed, binary values
$\hat{v}$
Visible unit $\hat{V}^k$:
Multinomial-distributed, sampled D times, non-negative, integer values
3 1 2 2

Figure 2.19: Weight sharing in the Replicated Softmax model: weights w_{ij}^k are restricted to w_j^k and the binary dictionary matrix V_i^k is aggregated to $\hat{V}^k$ by summation, resulting in a word counts vector for each document. See in color for better visualization.

Note that the notation in Eq. 2.90 sticks to the original proposition by Salakhutdinov. However, the weight sharing is not visibly reflected. It becomes clearer, if either one introduces additional constraints $W_{ij}^k = W_{\tilde{i}j}^k \forall i, \tilde{i}, j, k$ or one removes the index i from W. The latter one being less redundant and more elegant, of course.

The new RBM takes the $\hat{V}^k = \sum_{i=1}^{D} V_i^k$ as input and is now an RBM with a 'single' weight matrix W_j^k instead of i many weight matrices. The energy function now assumes the form:

$$E(\hat{V}, h) = -\sum_{j=1}^{F}\sum_{k=1}^{K} h_j W_j^k \hat{V}^k - \sum_{k=1}^{K} \hat{V}^k a^k - D\sum_{j=1}^{F} h_j b_j \tag{2.94}$$

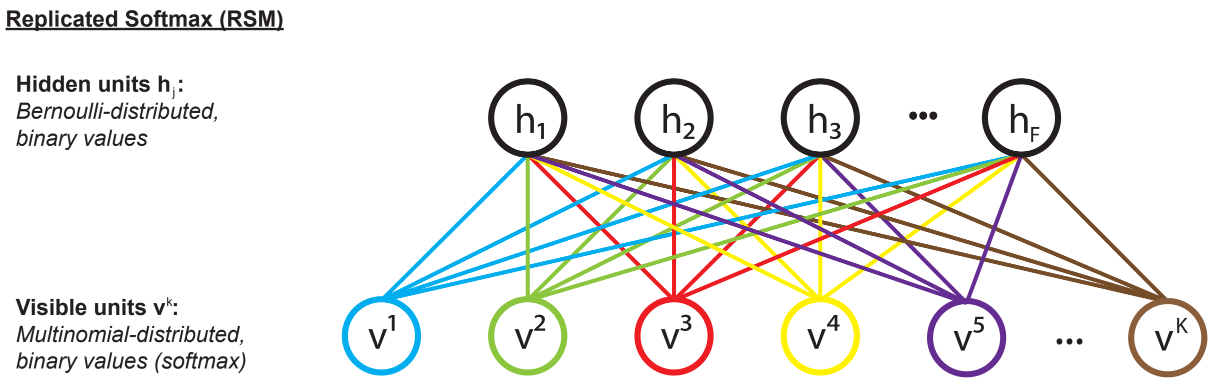
Replicated Softmax (RSM)
Hidden units h_j: Bernoulli-distributed, binary values
Visible units v^k: Multinomial-distributed, binary values (softmax)
h_1 h_2 h_3 ... h_F
v^1 v^2 v^3 v^4 v^5 ... v^K

Figure 2.20: Architecture of the Replicated Softmax model. See in color for better visualization.

For this new energy function sampling is achieved by:

$$p(h_j = 1|\hat{V}) = \sigma\left(Db_j + \sum_{k=1}^{K} W_j^k \hat{V}^k\right) \tag{2.95}$$

and

$$p(\hat{V}^k = 1|h) = \frac{\exp\left(a^k + \sum_{j=1}^{F} h_j W_j^k\right)}{\sum_{q=1}^{K} \exp\left(a^q + \sum_{j=1}^{F} h_j W_j^q\right)} \tag{2.96}$$

where $\hat{V}^k$ is sampled from D times. This is called the Replicated Softmax (RSM) model by the authors and can also be interpreted as K (keywords many) Restricted Boltzmann Machines that share the hidden units, see also Figure 2.20. Note that the biases for the hidden variables are multiplied by D, i.e. the document length. Due to this scaling, it is easy to cope with documents of different lengths. Of course, Contrastive Divergence learning can be applied to train this model.

It is also of interest, how this model differs from the Constraint Poisson model. One difference is where the scaling to document length takes place: the Constrained Poisson model scales the Poisson rates, whereas the Replicated Softmax model scales the hidden biases. The other difference is sampling: Given one uses sampling for the training, then the Replicated Softmax samples D times from *one* multinomial variable. In contrast to that the Constrained Poisson model samples from *keywords many* different Poisson distributions. Therefore, the Replicated Softmax model is guaranteed to sample the correct number of keywords in the document, whereas the Constrained Poisson model guarantees this solely in expectancy value.[14] However, as Salakhutdinov and Hinton already state, except for the sampling and scaling the two models are equivalent.

[14]Email Communication with Jan Schlüter. Thanks!

2.7.8 Dual Wing Harmonium (DWH)

The EFH model was extended by Xing et al. [62] to support two different members of the exponential family on the visible layer, i.e. two wings of input units, as illustrated in Figure 2.21. They call this model Dual Wing Harmonium (DWH) and use it for video classification.

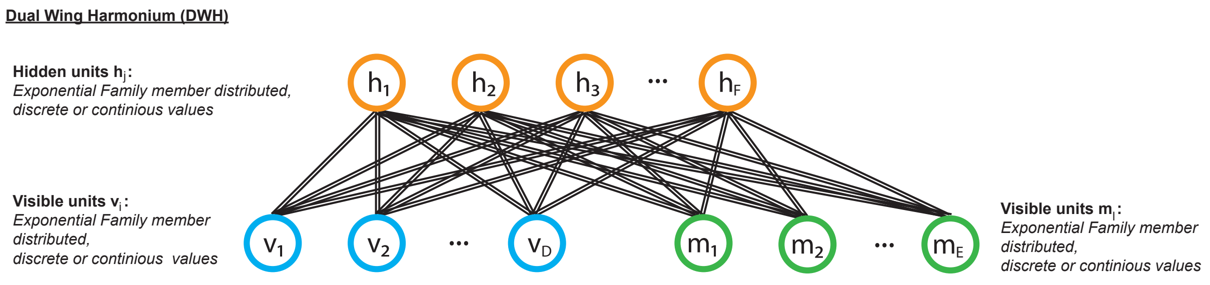
Dual Wing Harmonium (DWH)
Hidden units h_j:
Exponential Family member distributed, discrete or continious values
Visible units v_i:
Exponential Family member distributed, discrete or continious values
Visible units m_l:
Exponential Family member distributed, discrete or continious values
h1
h2
h3
...
hF
v1
v2
...
vD
m1
m2
...
mE

Figure 2.21: Architecture of a general Dual Wing Harmonium.

The energy function of the DWH is a straightforward extension of EFH's energy function:

$$E(v,m,h) \propto \exp\left(\sum_{ia} f_{ia}(v_i) + \sum_{lb} \eta_{lb} g_{jb}(m_l) + \sum_{jc} \lambda_{jc} e_{jc}(h_j) \right. \tag{2.97}$$

$$\left. + \sum_{iajc} W_{ia}^{jc} f_{ia}(x_i) e_{jc}(h_j) + \sum_{ljbc} U_{lb}^{jc} g_{lb}(m_l) e_{jc}(h_j) \right) \tag{2.98}$$

with two coupling matrices W_{ia}^{jc} and U_{lb}^{jc}, as well as two types of visible random variables v_i and m_l. Analogous to the EFH, model the $f_{ia}(v_i)$, $g_{jb}(m_l)$ and $e_{jc}(h_j)$ are the sufficient statistics of v, z and h. With A_i, B_l and C_j being the log-partition functions and θ_{ia}, η_{lb} and λ_{jc} being the natural parameters, the conditional distributions factorize as follows:

$$p(v|h) = \frac{p(v,h)}{p(h)} \propto \prod_{i=1}^{D} \exp\left(\sum_a \hat{\theta}_{ia} f_{ia}(v_i) - A_i\left(\{\hat{\theta}_{ia}\}\right)\right) \tag{2.99}$$

with $\hat{\theta}_{ia} = \theta_{ia} + \sum_{jc} W_{ia}^{jc} e_{jc}(h_j)$.

$$p(m|h) = \frac{p(m,h)}{p(h)} \propto \prod_{l=1}^{E} \exp\left(\sum_b \hat{\eta}_{lb} f_{lb}(m_l) - B_l\left(\{\hat{\eta}_{lb}\}\right)\right) \tag{2.100}$$

with $\hat{\eta}_{lb} = \eta_{lb} + \sum_{jc} U_{lb}^{jc} e_{jc}(h_j)$.

$$p(h|v,m) = \frac{p(v,m,h)}{p(v,m)} \propto \prod_{j=1}^{F} \exp\left(\sum_c \hat{\lambda}_{jc} f_{jc}(h_j) - C_j\left(\{\hat{\lambda}_{jc}\}\right)\right) \tag{2.101}$$

with $\hat{\lambda}_{jc} = \lambda_{jc} + W_{ia}^{jc} f_{ia}(x_i) + U_{lb}^{jc} g_{lb}(m_l)$.

The proof that this model factorizes into conditionally independent terms is straightforward, since the energy-function simply combines the two different types of variables additively. Consequently, CD learning is applicable by simply performing CD learning as usual, but sequentially for the two types of visible units.

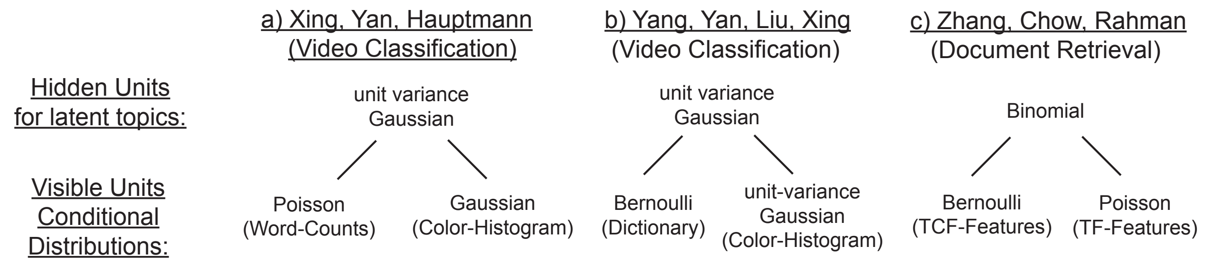

Figure 2.22: Different derived Dual Wing Harmonium models for video classification and document retrieval.

Xing et al. [62] use color-histograms extracted from video data modeling one wing of Gaussian visible units (learned variance), as well as annotation data, in particular word-counts w.r.t. a predefined dictionary on the other wing, modeled with conditional Poisson distributions. Finally, the hidden variables represent the influence of different latent topics, modeled by unit-variance Gaussian units.

Yang et al. [63] use a very similar approach as Xing et al., but use Bernoulli units for modeling the annotation data, which represents the presence or absence of pre-defined keywords. Additionally, they extend their DWH by adding another layer for classification on top of the DWH and call this Hierarchical Harmonium.

Zhang et al. [65] pick up on the idea of the DWH and used it for information retrieval, where they use *tf* and *tcf* features of documents, i.e. one wing is fed with the common term frequency feature (tf, i.e. word counts) modeled by Poisson rates. Additionally the second wing is fed with what they call **term connection frequency (tcf)** modeled with Bernoulli units. The tcf feature is an attempt to model the neighborhood relationships of keywords in the documents, i.e. relaxing the bag-of-word assumption. The hidden variables - representing the latent topics - are modeled with conditional Binomial distributions. Figure 2.22 gives an overview of the different applications of derived Dual Wing Harmoniums.

2.7.9 Practical issues of Contrastive Divergence learning

Hinton [16] presented several guidelines for training Restricted Boltzmann Machines with Contrastive Divergence. Instead of real sampling, sometimes the unit probabilities are employed in CD-k learning in order to reduce sampling noise. For binary units with CD-1, however, Hinton strongly suggests to use samples rather than unit probabilities.

Hinton also suggest to initialize the weight matrix by drawing samples from a zero-mean Gaussian random variable with standard deviation 0.01. The hidden variables biases should be initialized with 0 or a high negative value e.g. -4, where the latter case is a way to encourage sparse hidden activities. Hinton asserts (for the standard RBM), it is often helpful to initialize the visible units biases with $\log(r_i/(1-r_i))$, where r_i is the frequency of visible unit i being on. It is also possible to transform the input to the log-domain by applying the input transformation $\log(1+v_i)$, used e.g. by Salakhutdinov and Hinton [47].

For classification, the minibatch size should be about the number of target classes (e.g. 10) and each minibatch should contain a record from each class. The learning rate should be chosen such that the weight updates have magnitudes about 10^2 to 10^4 smaller than the weights. However, if non-binary units are involved, the learning rate should be chosen smaller to fight numerical stability issues, e.g. for Binomial units and especially for Gaussian units. One is also advised to divide the learning rate by the size of the minibatch, and therewith making the learning rate independent of the minibatch size. The momentum rate, cf. Appendix B, is advised to initially start with a value of 0.5 and over training progression slowly increases up to 0.9.

The learning can be monitored using the reconstruction error. However it is a proxy and not necessarily a good measure of performance. Therefore, Hinton suggests to use it carefully.

3 Own work

This Chapter covers our work, in particular the experiments using Neural Networks with Replicated Softmax input layers, modified error backpropagation and the DualRSM model. For convenient description of the experiments, we introduce the following abbreviations:

- RSM: Replicated Softmax layer
- RBM: Restricted Boltzmann Machine layer
- SIG: Multiple Logistic Regressions output layer
- SM: softmax output layer
- NN: standard Feed-forward Neural Network
- BP: error backpropagation
- K: dictionary size
- H: number of hidden units
- E: number of epochs
- L: learning rate
- M: momentum rate
- MBS: minibatch size

Throughout all experiments, Contrastive Divergence and error backpropagation use an effective learning rate calculated as learning rate divided by minibatch size. For all Tables presenting classification rates, either the percentage of correctly classified documents or (mean) Average Precision rates, the rates were cut off after 4 decimal digits. Note that equivalent RSM models are only trained once and weights resulting from a varying amount of pre-training epochs are obtained from one single pre-training run.

3.1 Evaluation of the Replicated Softmax model for document retrieval on the 20 Newsgroups dataset

In this Section, we attempt to reproduce the document retrieval results from the original Replicated Softmax (RSM) paper presented by Salakhutdinov and Hinton [47] on the 20 Newsgroups dataset. We choose this dataset, since the authors present recall precision curves for it, as well as its corpus size is comparably small but sufficient for our purposes. Additionally to building a

	Records	Minimum	Maximum	Average	Standard Deviation
Train	11314	5	12305	85.16	278.98
Test	7531	2	3387	75.47	131.86
Total	18845	2	12305	81.29	231.72

Table 3.1: 20 Newsgroups dataset statistics: number of Usenet articles and to the right of the double line minimum, maximum, mean and standard deviation of the amount of keywords present using a dictionary of the $K = 1998$ most frequent words.

dictionary based on the 2000 most frequent words like the authors, we can report significantly better results on a dictionary computed by highest information gain.

We use a setting similar to the original paper by Salakhutdinov and Hinton. The first step is to derive a dictionary from the 18845 Usenet articles collected in the 20 Newsgroups corpus. The **information gain (Infogain)** calculates how much information a word provides for classifying the respective document correctly. Thus, the words contributing most (highest information gain) are selected as dictionary keywords, see Manning et al. [32] for a detailed explanation. Using **Rainbow** [1] one can select those K keywords with the highest information gain or those that occur at least C times, i.e. selecting the 2000 most frequent words as keywords is not directly possible. Additionally, common stop-words like 'the', 'we' or 'that' are easily removed and similar words reduced to the same root word (stemming) - a common preprocessing in information retrieval. Selecting words that occur at least $C = 246$ times as keywords results in a dictionary of size $K = 2006$ and choosing $C = 247$ leads to a dictionary size of $K = 1998$.

The 6 most frequent words in the 2006 keywords dictionary are 'ax' (62551), 'subject' (20379), 'lines' (19673), 'organization' (18708), 'writes' (13344) and 'article' (12275) with occurrence counts denoted in brackets. The large occurrence of the term 'ax' results from one Usenet entry containing a picture encoded in ASCII. However, by application of the '–append-to-stoplist' command line switch, the 6 most frequent words can be removed from the dictionary of size $K = 2006$, thus resulting in a dictionary of size $K = 2000$. This might appear as a minor technical detail at first glance, but due to the dramatic gap of maximal occurrence counts: 625551 versus 9966 this has a strong impact on the training process of the Replicated Softmax model. Nevertheless, the authors report statistics (mean document length of approximately 50 keywords along with a standard deviation estimate of about 70), which we are not able to reproduce, cf. Table 3.2 and Table 3.1. Even repeated removal of most frequent words (that contribute most to the highest values to mean and standard deviation) does not lead to the reported results.

Despite being unable to employ equivalent dictionaries as the authors, our **recall precision curves** (RPCs) reinforce their results. The RPCs are calculated with the following definition of recall and precision:

$$\text{recall} = \frac{\#\text{correctly retrieved documents}}{\#\text{relevant documents in corpus}} \tag{3.1}$$

[1]`http://www.cs.umass.edu/~mccallum/bow/rainbow/`

	Occurrences $K = 2000$				Infogain $K = 2000$			
	Min	Max	Avg	StD	Min	Max	Avg	StD
Train	2	3162	75.278	148.697	2	2748	61.730	108.561
Test	0	3419	71.085	131.497	0	2935	59.891	103.116
Total	0	3419	73.602	142.084	0	2935	60.995	106.420

Table 3.2: 20 Newsgroups word counts statistics: keyword statistics for dictionaries built by selecting the 2000 most frequent words or those 2000 words with highest information gain.

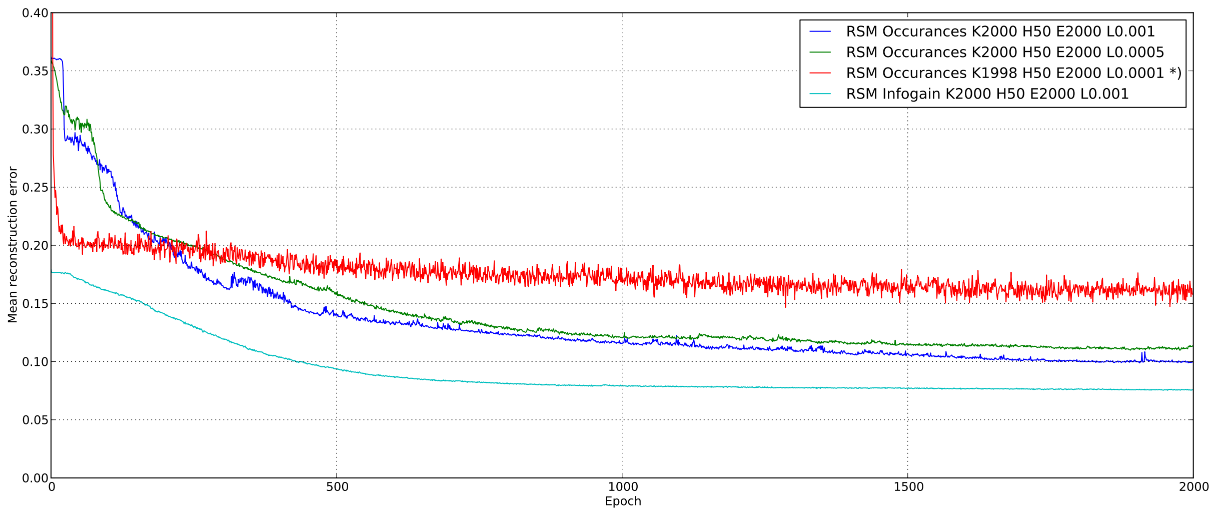

Figure 3.1: Mean reconstruction error by epochs. *) indicates a scaling along the vertical axis of the curve for convenient qualitative comparison (multiplied with 0.35). See in color for better visualization.

$$\text{precision} = \frac{\#\text{correctly retrieved documents}}{\#\text{retrieved documents}} \tag{3.2}$$

as defined e.g. by Gehler et al. [15].

Analogue to Salakhutdinov and Hinton we use a query record from the test set to retrieve similar documents from the train set, where similarity is carried out through the **cosine similarity** measure that is defined for two vectors u_i and u_j as follows:

$$\text{sim}(u_i, u_j) = \frac{u_i u_j^{\text{T}}}{\|u_i\|\|u_j\|} \tag{3.3}$$

see e.g. Manning et al. [32]. The labels are merely used to verify correctness of the model output, i.e. documents are similar, when they stem from the same class and for each test query we only receive the single most similar train document.

The authors do not report the learning rates they used, but they report to use 100,000 parameter updates for this dataset with a minibatch size of 100. We also do not apply the input transformation $\log(1+w_i)$ on word count w_i that is reported by the authors to give slightly better results.

The following experiments mainly attempt to find reasonable values for the hyperparameters of the RSM training, in particular learning rate and number of training epochs for the three different dictionaries described above. In this Section, we use a minibatch size of 100 and a constant momentum rate of 0.9. Weight matrix and both biases are initialized by drawing from a Gaussian distribution and multiplying the samples with 0.001. Moreover, we first permute the train records and then select as many as possible such that the total number of training records is dividable by the minibatch size without remainder.

Experiment 3.1.1: Learning rate.
Looking already forward to the Deep Belief Networks, we would like to know a reasonable learning rate and the amount of epochs to train the RSM input layer. For all input data, the learning rate needs to be below or equal to a value of 0.001 such that the reconstruction error (a proxy only, sure) drops smoothly over the complete training phase. As displayed in Figure 3.1 a learning rate of 0.001 works best for the information gain dictionary. For the most frequent occurrences dictionaries, a learning rate of 0.001 is still a bit aggressive, but leads for the $K = 1998$ dictionary to results comparable to the authors after approximately $140,000$ parameter updates, see Figure 3.4 for the $K = 1998$ recall precision curve.

Experiment 3.1.2: Number of training epochs.
For a fixed learning rate of 0.001, Figure 3.2 shows that the number of epochs the RSM is trained has crucial impact on the quality of the results. For the $K = 2000$ most frequent words dictionary the best results are obtained after 750 to 1250 training epochs, whereas for the highest information gain dictionary, the best result emerges after 500 training epochs.

Experiment 3.1.3: Robustness of the recall precision curves.
An important question is how robust these results for document retrieval are. Therefore, we trained 4 RSM models with the same parameter configuration. As Figure 3.3 illustrates, the recall precision curves vary strongly to the left of a recall value of 10^{-2}, but are rather stable right to it. Nevertheless, there is one outlier that departs for about 6% at the recall value of 10^{-2}.

Experiment 3.1.4: Dictionaries based on most frequent words versus highest information gain.
Figure 3.4 shows that RSM models trained on the dictionary based on $K = 2000$ keywords selected by highest information gain outperform the dictionary based on the $K = 2000$ most frequent words for up to 40%. The dictionary based on information gain outperforms the dictionary based on most frequent occurrences with $K = 1998$ keywords for up to 5 to 10%, but was trained for only 500 epochs rather than 2000. We do not have an explanation why the $K = 1998$ most frequent words dictionary performs so much better than the $K = 2000$ most frequent words dictionary.

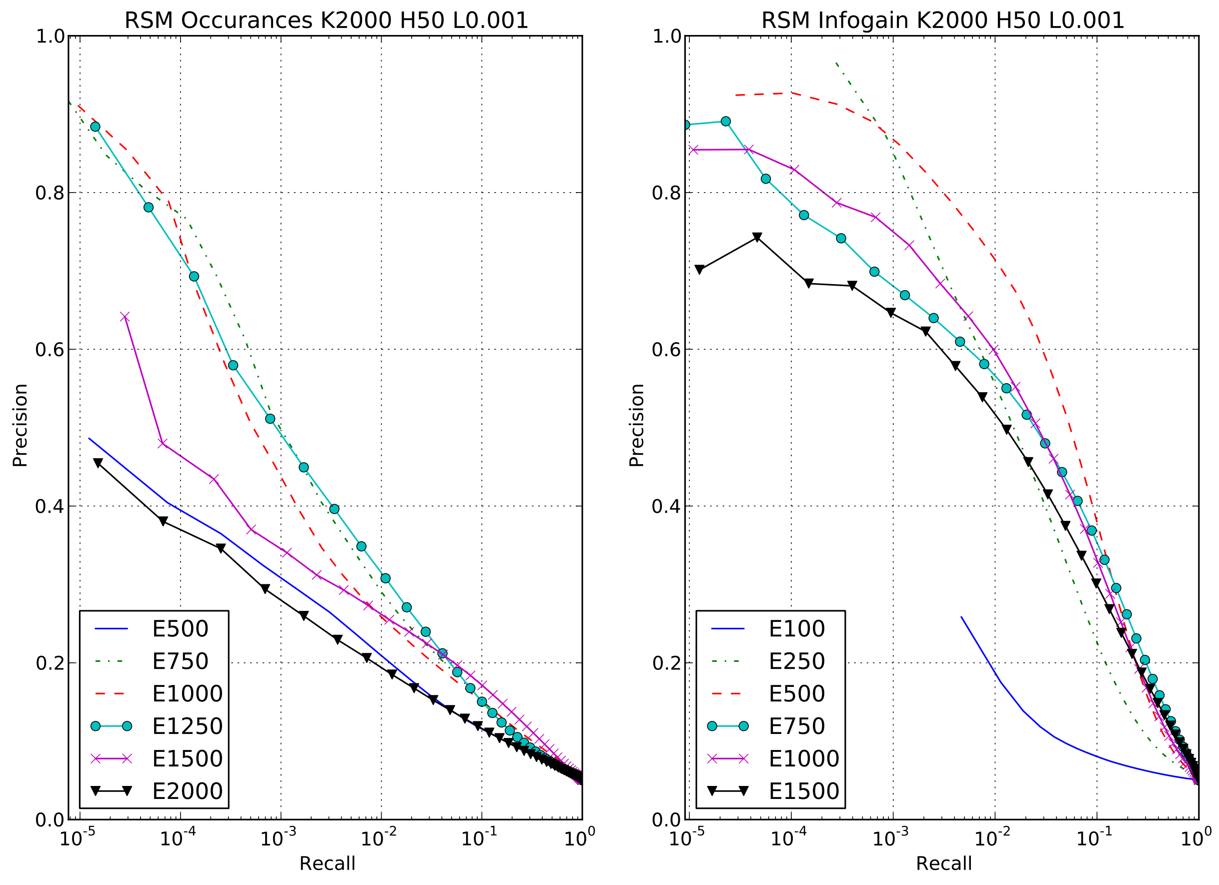

Figure 3.2: Recall precision curves for a varying number of epochs for a dictionary based on a) the 2000 most frequent words and b) on the 2000 words with highest information gain. The relevant part of the graph is at and to the right of the recall value 10^{-2}.

Experiment 3.1.5: Permuted dictionaries.
Due to the conditional independence of the visible random variables a permuted dictionary should not lead to qualitatively different results and the two recall precision curves (indicated by *) in Figure 3.4 confirms this. Hence, the two curves also serve as another test for robustness and reinforce the results of Experiment 3.1.3.

3.2 Neural Networks with Replicated Softmax input layers, modified error backpropagation and training details

A Deep Belief Network usually contains Restricted Boltzmann Machine layers as building blocks. However, depending on the desired output, a softmax (1-of-K coding scheme), logistic sigmoids (multiple binary classification) or identity output layer can be employed. Likewise, a RSM model can constitute the input layer and the hidden units output is fed into the succeeding layers. First, the RSM layer is pre-trained as described in Subsection 2.7.7. Secondly, the error backpropa-

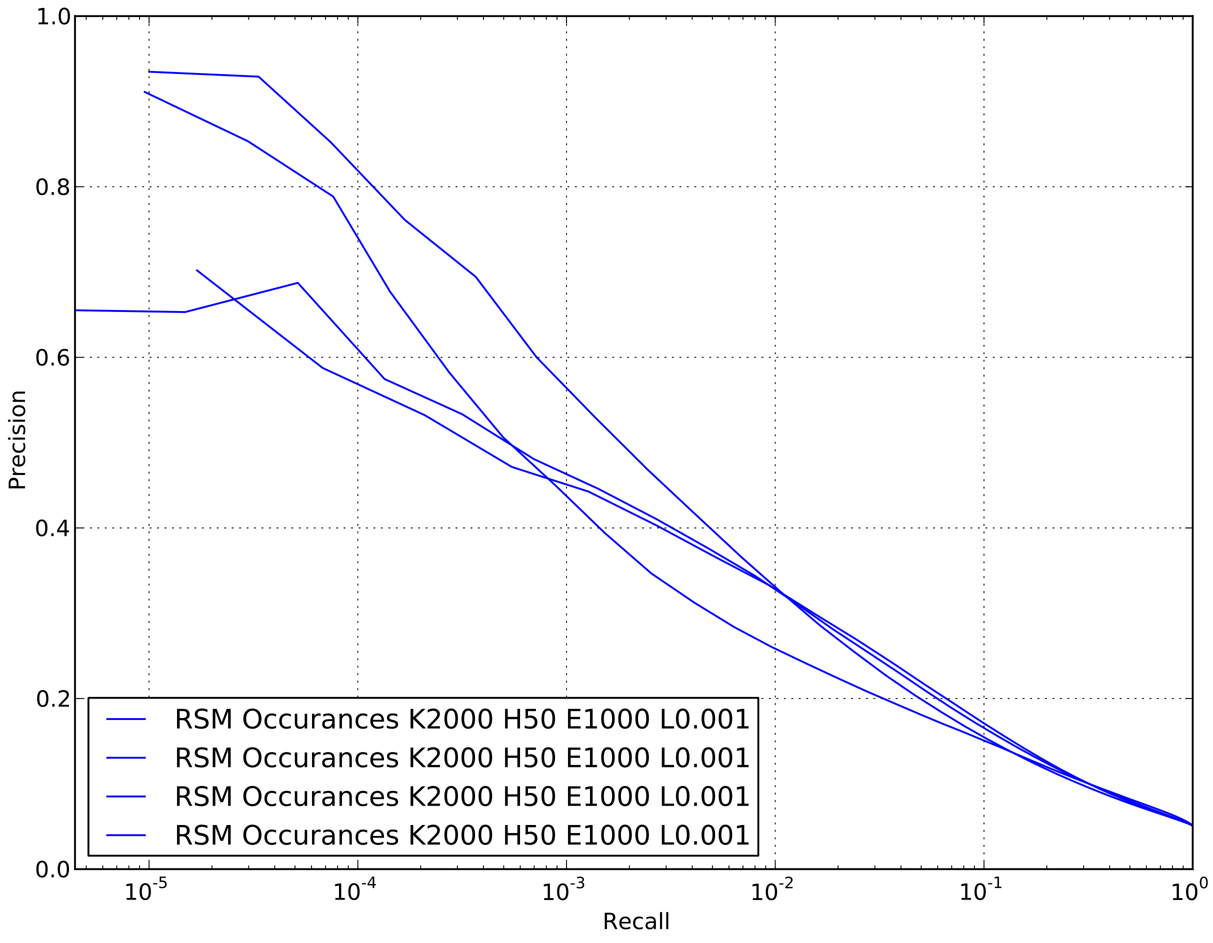

Figure 3.3: Evaluation of the robustness of the results by training four equivalent RSM models. The recall precision curves vary strongly left to a recall of 10^{-2} Note that the initialization of the weights is non-deterministic.

gation forward pass is carried out by calculating the hidden unit probabilities according to Eq. 2.95. Thirdly, the error backpropagation rule for the RSM layer needs to be modified, due to the scaling of the hidden biases by the document length D. This can be thought of by clamping units x_0 in Figure 2.8 to values D instead of 1. We call this the **modified error backpropagation** algorithm. Moreover, due to the scaling of the hidden biases, training a RSM model requires usually a pre-training learning rate of about 10^{-2} to 10^{-4} smaller than for a normal RBM. This also applies to the error backpropagation algorithm if a RSM layer is involved. The problem is worse for a logistic sigmoids output layer than for a softmax output layer. Because of the small learning rate, the RSM model needs to be pre-trained for a comparably large amount of epochs.

While the implementation for training a standard Restricted Boltzmann Machine and error backpropagation in a Neural Network is easily parallelized on the GPU using Gnumpy [54], my parallelized implementation of the Replicated Softmax model is slower than my CPU implemen-

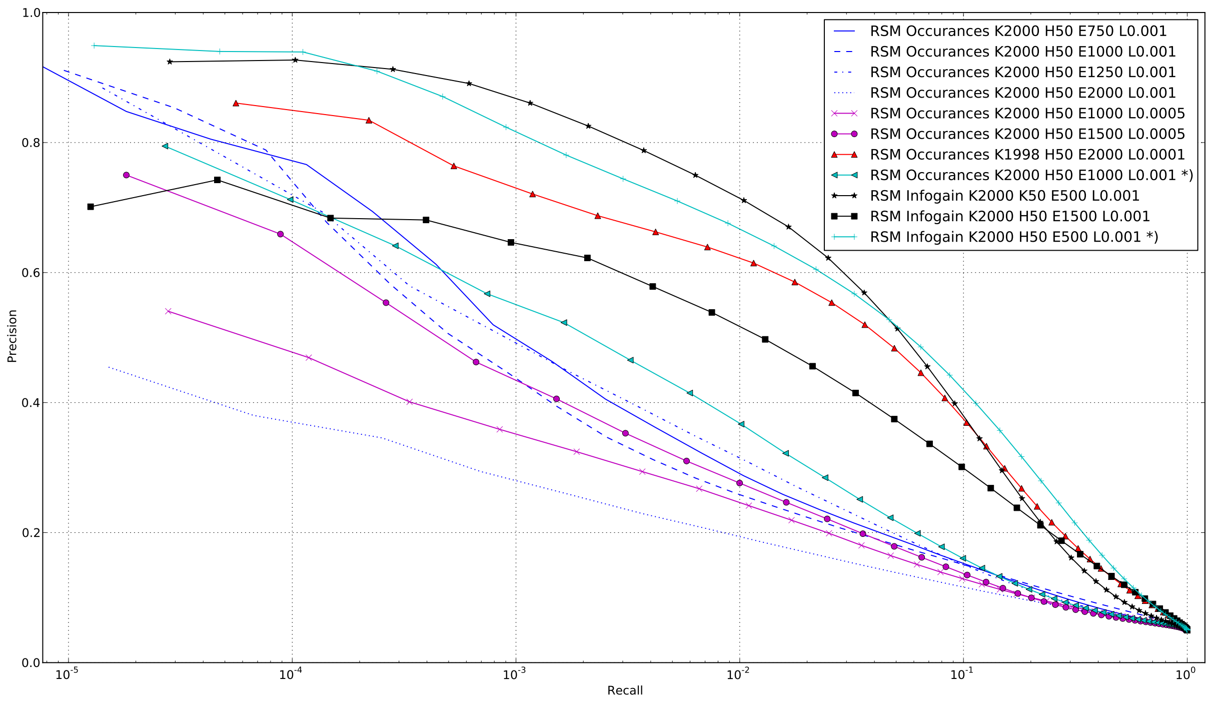

Figure 3.4: Comparison of the recall precision curves obtained from different dictionaries and different training hyperparameters, *) indicates a permuted dictionary. Top performers are the dictionaries based on $K = 1998$ most frequent words and highest information gain ($K = 2000$), trained for $E = 2000$ and $E = 500$ epochs.

tation, due to the sampling from the multinomial random variable. It would require a parallel version of cumulative sum in Cudamat [36], which Gnumpy is based on, or any other efficiently parallelized version of sampling from multinomial variables.

3.3 Document classification using Neural Networks with Replicated Softmax input layers on the 20 Newsgroups dataset

This Section covers the training of Neural Networks with a Replicated Softmax input and softmax output layers, in order to carry out classification on two different dictionaries: the $K = 2000$ most frequent words and the $K = 2000$ words selected by highest information gain, as described in Section 3.1. The ground truth data is used for modified error backpropagation in the Neural Network after unsupervised pre-training of the layers and for the identification of true positives. There are many hyperparameters for a Neural Network and the following experiments attempt to find reasonable values, e.g. learning rates, number of hidden values and number of weight layers. Analogue to Section 3.1, the dictionary based on highest information gain outperforms the dictionary based on the most frequent words. Throughout this Section, the RSM layer is pre-trained with a minibatch size of 128 and a constant momentum rate of 0.9. For all Exper-

Occurrences $K = 2000$, SM.E20			
RSM.H \RSM.L, RSM.E	0.0001, 1000	0.0005, 1000	0.0005, 2000
256	75.39 (134)	75.64 (68)	75.90 (158)
512	75.24 (133)	76.33 (197)	76.44 (188)
1024	74.15 (234)	76.49 (110)	76.50 (233)
2048	75.45 (143)	75.91 (248)	**76.57 (68)**
Occurrences $K = 2000$, SM.E50			
RSM.H \RSM.L, RSM.E	0.0001, 1000	0.0005, 1000	0.0005, 2000
256	75.30 (240)	75.56 (149)	75.88 (121)
512	75.61 (199)	76.25 (237)	76.17 (95)
1024	74.91 (150)	76.01 (219)	**76.89 (133)**
2048	75.86 (166)	76.27 (56)	76.60 (197)

Table 3.3: Classification rates for a **RSM->SM** model using the $K = 2000$ **most frequent words** dictionary for different learning rates of the RSM layer (RSM.L), varying number of epochs for the RSM layer (RSM.E) and the softmax output layer (SM.E) and various number of hidden units (RSM.H). The integers in brackets indicate the error backpropagation early stopping epoch.

iments, 250 modified error backpropagation epochs were carried out and the one epoch having the best result on the test set selected (early stopping). The RSM weights are initialized as described in Section 3.1.

Experiment 3.3.1: RSM->SM models.
In this first Experiment, we train a simple Neural Network with a RSM input layer and a successive softmax output layer. Both layers are pre-trained and subsequently the complete model is fine-tuned using error backpropagation (see Section 3.2 for details on how to adapt the error backpropagation algorithm to cope with RSM input layers). Here the parameters of interest are the learning rate and the number of epochs for pre-training the RSM layer, the number of hidden units, the number of pre-training epochs for the softmax output layer and the number of error backpropagation epochs.

Table 3.3 presents the results on the dictionary based on the $K = 2000$ most frequent words and Table 3.4 for the dictionary based on the $K = 2000$ keywords selected by highest information gain. In general, training a network with a softmax output layer is quite robust. Hence, training the RSM longer, with a slightly aggressive learning rate and more hidden units typically gives better results. However, as expected the softmax output layer should only be trained for about 20 to 30 epochs, such that the error backpropagation algorithm does not get stuck in poor local optima. With an error backpropagation learning rate of 0.05 fine-tuning for up to 200 epochs and applying early stopping proves useful. For the occurrences dictionary, the RSM learning rate of 0.0005 is quite aggressive in contrast to the information gain dictionary where it is much less aggressive as the error backpropagation starts on the latter on a much higher level.

Infogain $K = 2000$ SM.E20			
RSM.H \RSM.L, RSM.E	0.0001, 500	0.0005, 500	0.0005, 1000
256	77.92 (248)	79.63 (119)	79.76 (211)
512	78.13 (231)	79.26 (132)	**80.34 (189)**
1024	78.16 (107)	79.10 (221)	78.82 (246)
2048	77.84 (213)	78.58 (58)	79.83 (178)
Infogain $K = 2000$, SM.E50			
RSM.H \RSM.L, RSM.E	0.0001, 500	0.0005, 500	0.0005, 1000
256	78.00 (163)	79.60 (239)	79.35 (221)
512	78.46 (239)	79.41 (123)	79.00 (102)
1024	78.12 (198)	79.20 (212)	**79.86 (81)**
2048	77.08 (131)	78.71 (161)	79.8087 (75)

Table 3.4: Classification rates for a **RSM->SM** model using the highest **information gain** dictionary of size $K = 2000$ for different learning rates of the RSM layer (RSM.L), varying number of epochs for the RSM layer (RSM.E) and the softmax output layer (SM.E) and various number of hidden units (RSM.H). The integers in brackets indicate the error backpropagation early stopping epoch.

Occurrences		
RBM.E	RBM.H256, RSM.H256	RBM.H2048, RSM.H2048
50	76.14 (146)	77.27 (232)
100	76.57 (243)	77.03 (110)
150	76.15 (182)	77.58 (154)
200	**76.60 (182)**	76.40 (216)
250	76.58 (116)	77.43 (243)
300	75.82 (249)	77.31 (241)

Table 3.5: Classification rates for a **RSM->RBM->SM** model using the $K = 2000$ **most frequent words** dictionary for a varying amount of RBM layer pre-training epochs (**RBM.E**) for a very small RSM and RBM layer, as wells as for very large RSM and RBM layers. The integers in brackets indicate the error backpropagation early stopping epoch.

Experiment 3.3.2: RSM->RBM->SM models and training epochs.
Here, we train a model with one RBM layer in between the Replicated Softmax input layer and the softmax output layer that is pre-trained with a standard RBM. We tried to determine a suitable number of pre-training epochs for the RBM layer first. As Table 3.5 shows for the occurrences dictionary, the impact of the number of epochs is low, hence we choose RBM.E=100 epochs in combination with a learning rate of RBM.L=0.1.

An important question is how many hidden units the RSM and RBM layer in a RSM->RBM->SM model should have, since this directly affects the expressional power of each layer. Tables 3.6 and 3.7 indicate the expected result that more hidden units provide better results, i.e. the lower triangle along the counter-diagonal yields in general better results than the upper triangle.

Occurrences (SM.E20, L.0005, E2000)					
RSM.H \RBM.H	256	512	1024	2048	4096
256	75.83 (84)	76.21 (122)	76.13 (170)	76.71 (142)	76.05 (142)
512	76.48 (172)	76.64 (154)	76.62 (242)	76.85 (146)	76.25 (249)
1024	77.00 (217)	76.95 (235)	76.60 (227)	77.27 (145)	77.20 (151)
2048	77.15 (170)	76.45 (248)	76.98 (147)	76.85 (76)	**77.34 (206)**

Table 3.6: Classification rates for **RSM->RBM->SM** models with **differently sized RSM and RBM layers** on the $K = 2000$ **most frequent words** dictionary. The integers in brackets indicate the error backpropagation early stopping epoch.

The results indicate that one RBM layer can improve the classification results for about one or two percent.

Infogain (SM.E20, BP.L0.0005, E1000)					
RSM.H \RBM.H	256	512	1024	2048	4096
256	79.53 (106)	79.91 (241)	79.66 (207)	80.10 (223)	79.93 (148)
512	80.41 (227)	80.50 (131)	80.73 (146)	80.42 (198)	80.67 (197)
1024	80.63 (178)	80.45 (228)	80.92 (199)	80.77 (105)	**81.30 (166)**
2048	80.30 (235)	80.44 (174)	80.71 (197)	81.19 (116)	80.54 (177)

Table 3.7: Classification rates for **RSM->RBM->SM** models with **differently sized RSM and RBM layers** on the dictionary bases selected by highest **information gain**. The integers in brackets indicate the error backpropagation early stopping epoch.

Experiment 3.3.3: Deep Belief Networks of varying depth.
A Deep Belief Network is usually capable of detecting higher order correlations. Therefore, we tried to train Deep Belief Networks with RSM input layers, SM output layers and multiple RBM layers. However, as Tables 3.8 and 3.9 show, multiple hidden layers do not improve upon classification performance.

Occurrences (SM.E20, RBM.H1024)			
RSM.H \RBM layers	2	3	4
256	76.09 (104)	76.03 (153)	76.34 (163)
512	76.54 (235)	76.02 (107)	75.17 (248)
1024	76.44 (140)	76.38 (236)	75.29 (223)
2048	**76.85 (244)**	76.65 (249)	75.39 (178)

Table 3.8: Classification rates on the **most frequent words** dictionary for a **varying amount of hidden layers**. The integers in brackets indicate the error backpropagation early stopping epoch.

Each hidden layer has equivalent many hidden units of the same size as the respective RSM input layer's hidden units. All RBM layers are trained for 100 epochs with learning rate 0.1.

Infogain (SM.E20, RBM.H1024)			
RSM.H \RBM layers	2	3	4
256	79.28 (140)	79.40 (292)	79.04 (300)
512	79.74 (176)	80.03 (183)	79.62 (269)
1024	80.17 (255)	**80.75 (299)**	79.63 (348)
2048	79.64 (149)	79.90 (312)	79.94 (195)

Table 3.9: Classification rates on the dictionary based on the highest **information gain** for a **varying amount of hidden layers**. The integers in brackets indicate the error backpropagation early stopping epoch.

Experiment 3.3.4: Other variants of Deep Belief Networks.
We tried to train a Deep Belief Network with one large hidden layer followed by layers with deceasing many hidden units, in particular the following model:
RSM.H1024->RBM.H4096->RBM.H1024->RBM.H256->SM.C20
However, this topology does not provide a better classification rate for the information gain dictionary with 80.23% in error backpropagation epoch 167.
Another option is to increase the number of hidden units for only one RBM layer:
RSM.H2048->RBM.6144->SM.C20
While increasing the number of hidden units often leads to better classification rates, cf. Experiment 3.2.2, utilizing 6144 hidden units cannot further improve upon this behavior. The result for this model on the information gain dictionary is 80.87% in error backpropagation epoch 160.

Moreover, we perform document classification analogue to the experiments in Tables 3.3 and 3.6 using the $K = 1998$ most frequent words dictionary with the RSM model trained for 2000 epochs with learning rate 0.0001. All results vary at most about 1% with the best absolute classification rate being 76.56% in modified error backpropagation epoch 249. In all experiments the modified error backpropagation is able to improve upon classification performance. The large number of 2000 pre-training epochs of the RSM model thus seems not sufficient for the classification setting.

In summary, the dictionary based on the $K = 2000$ words with highest information gain outperforms the dictionary based on $K = 1998$ and $K = 2000$ most frequent words for about 5%, which is consistent with the results of Section 3.1. Furthermore, in the case of a 1-of-K coding scheme, longer training, more hidden units and adding one RBM layer can slightly improve the classification results for about one to two percent. Lacoste-Julien et al. [26] report a *mis*classification rate of 25% on the 20 Newsgroups dataset with standard LDA and a *mis*classification rate of 20% using DiscLDA. Therefore, the RSM->SM model on the most frequent words dictionary is competitive to standard LDA and the RSM->SM model using a dictionary seleceted by highest information gain is competitive to the DiscLDA model.

3.4 Extraction of visual words and visual word counts

The first step in our image object classification pipeline is to obtain visual words. This first step is already crucial and low quality feature points and descriptors can lead to poor results.

We experimented with Lowe's original SIFT [29] implementation, VLFeat[2] by Vedaldi [56] and SIFT++, also authored by Vedaldi. Figure 3.5 illustrates that different SIFT implementations result in strongly varying descriptors and amounts of descriptors. Vedaldi's VLFeat was accessed by the provided binary as well as through a python wrapper, however, both return a lot of descriptors covering - for the human eye - non-interesting aspects of the respective images. In contrast to that Lowe's implementation (hard coded parameters) and SIFT++ with default parameters seem to provide reasonable results. We tried to obtain more distinctive descriptors by passing non-default threshold (set to 0.019) and edge-threshold (set to 4.0) parameters to SIFT++.

An interesting observation is depicted in Table 3.10, in particular that SIFT++ returns about twice as many descriptors in total than Lowe's SIFT implementation. The idea of the custom parameters for SIFT++ was to obtain less, but higher quality descriptors and indeed only delivers half the amount of descriptors than Lowe's SIFT implementation.

Moreover, we tried to collect descriptors using PCA-SIFT by Ke and Sukthankar [24]. The PCA-SIFT (k=36) implementation provided by the authors is shipped without an interest point detector. First, we tried to feed PCA-SIFT with the keypoints detected by SIFT++. Unfortunately, many resulting descriptors are broken, in detail: all 36 resulting elements contain 'inf'. Using keypoints detected by Lowe's implementation this problem diminishes, yet for another to arise: About 60 images in both, train and test set, PCA-SIFT ends up in a segmentation fault. Therefore, we removed those images and the respective targets for the remainder of our pipeline.

		# Descriptors	Min	Max	Avg	StD
SIFT++ default	Train	**6188387**	3	5241	1234.96	17.65
	Test	6080382	9	4882	1227.86	17.41
Lowe SIFT	Train	**4380138**	3	4185	874.10	17.82
	Test	4325806	10	4449	873.54	17.59
SIFT++ T0.019 E4	Train	**2268042**	2	3078	452.61	15.52
	Test	2242542	3	3186	452.94	15.64

Table 3.10: SIFT descriptor statistics obtained by different SIFT implementations and parameters.

The next step in the pipeline is to quantize the visual words into visual word counts. The 6 million SIFT descriptors gathered by the SIFT++ implementation on the training set barely fit into 8GB main memory, hence we collect 3 million SIFT descriptors from randomly chosen training images and then subsample 1 million descriptors, as performed similarly e.g. by Zhou et al. [66]. Next, we apply the Mini-batch k-Means algorithm on the one million subsampled descriptors. The distance calculation within a minibatch is easily parallelized on the GPU which greatly improves upon calculation time (about one hour computation time).
The true error is very expensive to calculate. Therefore, we observe a proxy error that is calculated easily along training: the mean Euclidean distance of all selected descriptors of a minibatch

[2]`http://www.vlfeat.org`

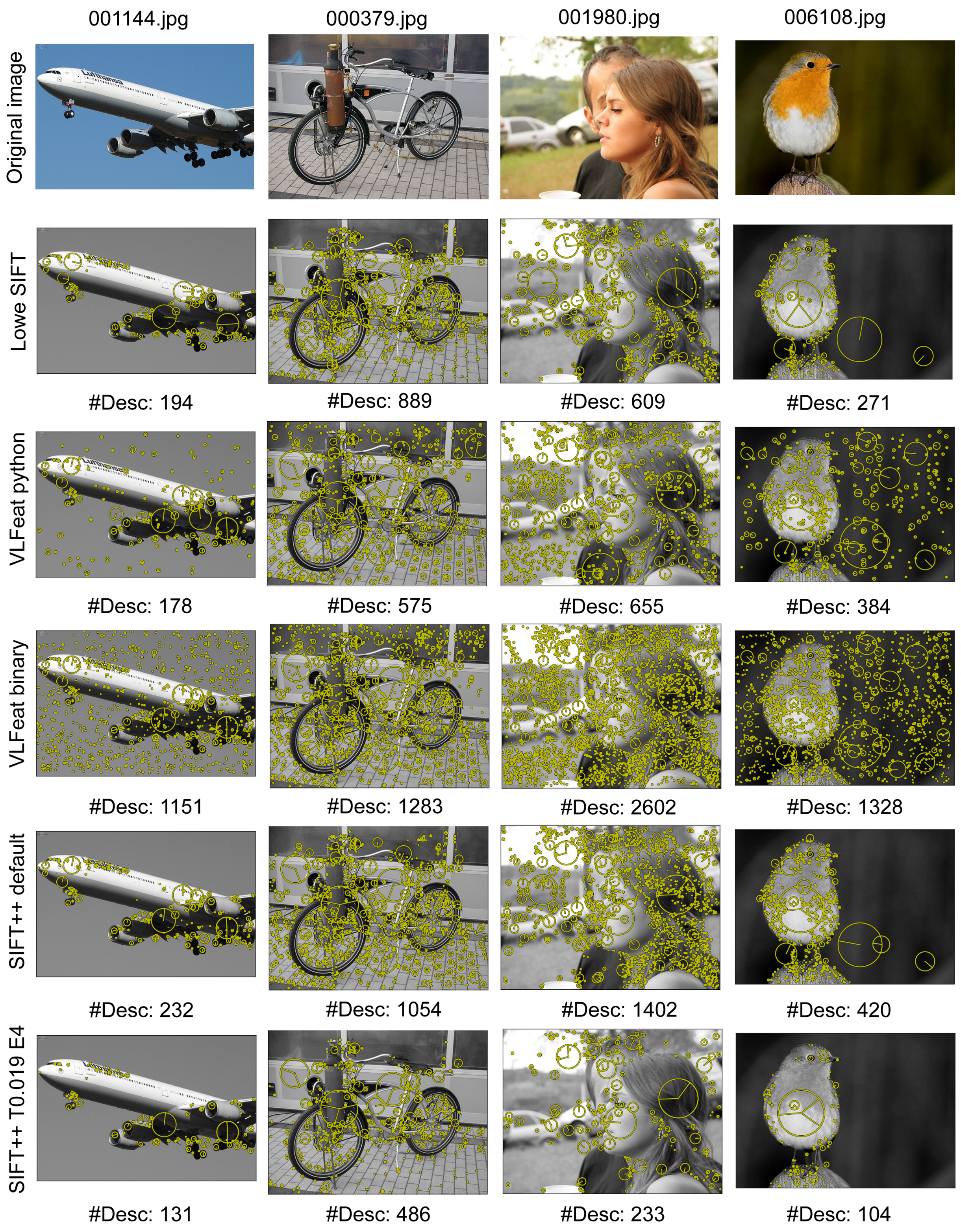

Figure 3.5: Visualization of all SIFT descriptors (counts below image) on different selected images, SIFT implementations (Lowe, Vedaldi's VLFeat binary, VLFeat python wrapper and SIFT++) and parameters (SIFT++ with default parameters and with threshold and edge-threshold parameters set to 0.019 and 4.0). See in color for better visualization.

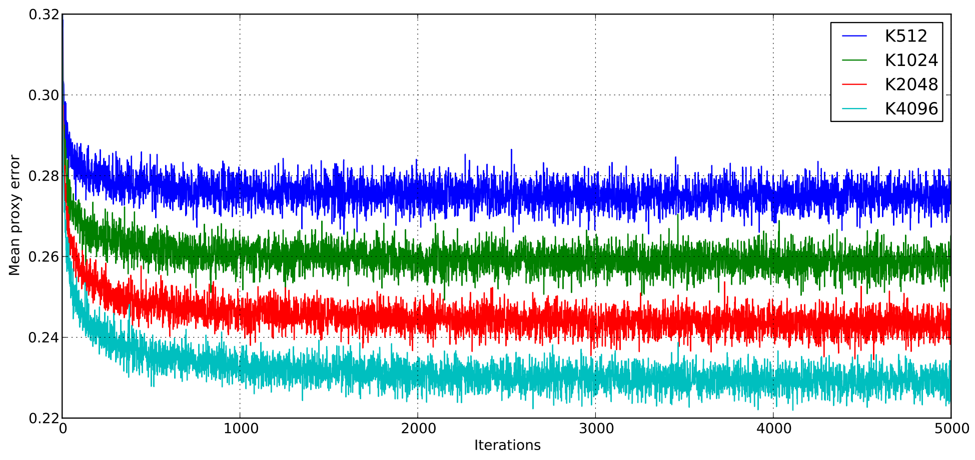

Figure 3.6: Progress of the mean Euclidean distance for different dictionary sizes K of all selected descriptors of a minibatch to their respective cluster centers (**mean proxy error**) over the first 5000 iterations of the **Mini-batch k-Means** algorithm run on the SIFT descriptors obtained with SIFT++ without kmeans++ initialization. See in color for better visualization.

to their respective cluster centers. As shown in Figure 3.6 the error decreases most in the first 500 iterations. We found that for our pipeline that 1000 iterations are not fully sufficient, but 10.000 or 50.0000 iterations do not lead to any significant difference. In the following 10.000 iterations are used.

We use a minibatch size of 1000 here. Eventually, a larger minibatch size could lead to better overall results of the pipeline. Jan Schlüter suggested to use a minibatch size eight times larger than the dictionary size.[3]

We also tried an initialization of **Mini-batch k-Means with kmeans++ (MBKM++)**. The proxy error appears slightly lower in the first 400 iterations, but leads to a bit worse absolute results in the overall pipeline and without parallelization is very costly in our pipeline.

3.5 Evaluation of Neural Networks with Replicated Softmax input layers on visual word counts

Given the visual word counts carried out in the previous Section 3.4, we now perform multilabel image object classification by training Neural Networks with a Replicated Softmax input and sigmoidal output layers **(RSM->SIG)**. Replicated Softmax and sigmoidal layers are pretrained and afterwards the merged model is fine-tuned afterwards using modified error backpropagation. The sigmoidal layer is a regular Feed-forward Neural Network layer with sigmoid activation functions and as many output units as target classes to predict. The output of a

[3]E-Mail communication with Jan Schlüter, thanks!

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
SIG tf	14.96 (920)	15.00 (987)	14.90 (912)	15.04 (997)
NN tf-idf	13.48 (194)	22.21 (58)	23.87 (277)	22.18 (484)
NN tf	27.16 (34)	26.13 (19)	25.14 (249)	23.76 (497)
RSM.E100	29.04 (250)	29.34 (196)	28.42 (240)	27.29 (250)
RSM.E250	**30.69 (35)**	31.81 (17)	31.66 (69)	30.37 (136)
RSM.E400	30.36 (217)	**32.26 (0)**	**31.91 (126)**	31.39 (0)
RSM.E500	30.59 (6)	32.18 (0)	31.77 (117)	**31.53 (0)**
RSM.E750	30.22 (33)	31.61 (10)	31.59 (127)	31.35 (0)
RSM.E1000	30.17 (2)	31.30 (6)	30.89 (145)	30.21 (0)

Table 3.11: Mean Average Precision classification rates on visual word counts obtained by **Lowe's SIFT** implementation and Mini-batch k-Means.

neuron in the sigmoidal layer is treated as the 'confidence' of the model that an object is present in an image, as described in the PASCAL VOC 2007 Documentation. The result is evaluated using class **Average Precision (AP)**, i.e. for each class recall and precision are calculated and the area under the recall precision curve is computed.[4] The final score **mean Average Precision (mAP)** is the non-weighted mean of all per-class Average Precision values. We do not remove **visual stop words**, since the results of Jiang et al. [21] suggest that the removal of visual stop words does not improve the performance in the bag-of-visual-words framework. It might however lead to a more efficient pipeline with only slightly less performance.

Throughout this Section we use a minibatch size MBS = 128 and we train the Replicated Softmax layers with a constant momentum rate of 0.9 for varying amounts of epochs with learning rate 0.0001. This small learning rate is necessary for smooth learning in terms of the reconstruction error. The sigmoidal output layers are pre-trained for 30 epochs with learning rate 0.1. The RSM weights are initialized as described in Section 3.1. All RSM-based models are fine-tuned using modified error backpropagation for 250 epochs and the best result on the test set is selected (early stopping epoch denoted in brackets).

We baseline against standard Feed-forward Neural Networks with sigmoidal output layers, trained solely with standard error backpropagation and fed with visual word counts (tf) and tf-idf values, the latter one stemming from document retrieval. The term frequency ($\mathrm{tf}_{k,i}$) equals the visual word counts. The document frequency (df_k) counts the number of documents in the corpus where keyword k is present. The inverse document frequency then is:

$$\mathrm{idf}_k = \log \frac{D}{\mathrm{df}_k} \tag{3.4}$$

where D is the total number of documents, i.e. words occurring in many documents are furnished with smaller weights. Afterwards, the tf-idf values are combined as the product of document-

[4]Documentation and development kit code: `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/VOCdevkit_08-Jun-2007.tar`, VOCdevkit/devkit_doc.pdf, VOCdevkit/VOCcode/VOCevalcls.m

wide idf and document-specific tf values:

$$\text{tf-idf}_{k,i} = \text{tf}_{k,i} \cdot \text{idf}_k \tag{3.5}$$

for keyword k and document i, see e.g. Manning et al. [32].

Experiment 3.5.1: RSM->SIG models on visual word counts obtained using different SIFT implementations and descriptors.
After experimenting a bit with all the hyperparameters in our pipeline we found that likewise to the classification task on the 20 Newsgroups dataset, the number of epochs the RSM layer is pre-trained, is crucial (for a fixed learning rate). Tables 3.11 to 3.14 now present the mAP classification rates for two-layer Neural Networks on different SIFT descriptors, obtained by Lowe's original SIFT implementation, PCA-SIFT(36) (from Lowe SIFT keypoints) and SIFT++ with default parameters, as well as SIFT++ with threshold and edge-threshold parameters set to 0.019 and 4.0. On each subsampled descriptor set we performed a clustering with $K = 512, 1024, 2048, 4096$ cluster centers with Mini-batch k-Means, in this Experiment without kmeans++ initialization. The corresponding RSM input layer has as many hidden units as cluster centers.
In this Experiment, the RSM-SIG models are fine-tuned with a learning rate of 0.0001 for 512 cluster centers and 0.00001 for the others.

The standard Feed-forward Neural Network (NN) is solely trained with standard error backpropagation for 500 epochs with learning rate 0.1 and equivalent to the RSM->SIG model fed with tf and tf-idf values. Moreover, we trained a single sigmoidal layer (SIG) directly fed with tf values and trained for 1000 epochs with learning rate 0.1.

Clearly, the number of pre-training epochs of the RSM layer has a strong impact. Usually, there is a number of pre-training epochs, where the modified error backpropagation algorithm can easily improve upon. After some more pre-training epochs, the best results are obtained directly after pre-training (indicated by early stopping epoch 0 in brackets, i.e. modified error backpropagation cannot improve upon pre-training). From that point on error backpropagation takes much more epochs to improve upon pre-training (if at all), i.e. the modified error backpropagation algorithm first has to rewind parts of the pre-training. However, if no pre-training is used at all, then the modified error backpropagation algorithm is very capable of improving the classification results over training progression, but of course does not reach the results reported here.

Interestingly, the SIFT implementation returning the largest number of descriptors performs best, while using descriptors obtained by custom SIFT++ parameters or PCA-SIFT(36) (where information is dropped) lead to quantitatively worse results. We observed that the modified error backpropagation algorithm does not perform as stable as in the softmax output layer case in Section 3.3. However, in my experience recall precision curves often have robustness problems. Also note that similar to Jiang et al. [21] using tf-idf values, the results are better for larger dictionaries than for smaller dictionaries.

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
SIG tf	14.00 (603)	13.53 (996)	15.10 (923)	15.09 (844)
NN tf-idf	13.94 (167)	19.69 (96)	22.82 (104)	21.37 (495)
NN tf	25.52 (31)	24.22 (32)	24.56 (108)	22.74 (500)
RSM.E100	26.83 (250)	26.29 (149)	25.46 (250)	24.53 (197)
RSM.E250	**28.61 (241)**	29.34 (134)	29.12 (144)	27.40 (112)
RSM.E400	28.51 (7)	**29.65 (0)**	**30.94 (0)**	28.78 (239)
RSM.E500	28.43 (229)	29.53 (22)	30.56 (0)	29.12 (250)
RSM.E750	28.35 (249)	29.41 (250)	29.49 (0)	**29.48 (0)**
RSM.E1000	28.45 (97)	28.99 (233)	28.71 (0)	28.91 (179)

Table 3.12: Mean Average Precision classification rates on visual word counts resulting from Ke and Sukthankar **PCA-SIFT(36)** on keypoints detected by Lowe's SIFT implementation and Mini-batch k-Means.

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
SIG tf	13.70 (296)	14.27 (476)	14.21 (691)	14.39 (978)
NN tf-idf	20.05 (72)	22.86 (48)	22.57 (96)	20.31 (477)
NN tf	24.44 (32)	23.67 (30)	23.47 (495)	21.94 (410)
RSM.E100	24.09 (249)	21.39 (250)	21.23 (250)	19.16 (250)
RSM.E250	26.59 (15)	25.53 (193)	24.73 (250)	22.49 (250)
RSM.E400	27.58 (0)	26.20 (227)	26.84 (63)	25.01 (250)
RSM.E500	27.67 (0)	26.62 (3)	27.04 (81)	26.25 (244)
RSM.E750	27.92 (6)	27.33 (41)	27.46 (0)	26.39 (0)
RSM.E1000	**28.03 (0)**	27.72 (51)	27.46 (106)	27.44 (0)
RSM.E1250	27.74 (8)	27.77 (250)	27.49 (25)	27.28 (0)
RSM.E1500	27.72 (6)	**27.86 (240)**	**27.61 (0)**	**27.58 (0)**
RSM.E1750	27.77 (9)	27.41 (91)	27.25 (0)	27.46 (0)
RSM.E2000	27.97 (2)	27.45 (34)	27.33 (0)	26.88 (0)

Table 3.13: Mean Average Precision classification rates on visual word counts obtained by **SIFT++ with threshold and edge-threshold parameters set to 0.019 and 4.0** and Mini-batch k-Means.

Experiment 3.5.2: RSM->SIG models on visual word counts gathered by SIFT++ with default parameters and Mini-batch k-Means initialized with kmeans++.
We choose SIFT++ with default parameters as best performer in the previous experiment and the single change here is to initialize Mini-batch k-Means with kmeans++ (MBKM++). Table 3.15 shows that the results are comparable. The results may be slightly more stable, nevertheless, we do not use it here in succeeding experiments, because without parallelization kmeans++ costs a lot of computation time.

Table 3.16 displays the class Average Precisions for the top performers of each Table so far (Tables 3.11 to 3.15).[5] It is very interesting that each SIFT implementation can perform very well at at least for one object class, except for PCA-SIFT, which actually really removes information from the descriptors. Worthwhile to note is the very good performance for the class *potted_plant* using SIFT++ with custom parameters. Nevertheless, SIFT++ with default parameters performs best in terms of mAP in both cases: with or without kmeans++ initialization.

Experiment 3.5.3: Varying the number of hidden units in the RSM layer in RSM->SIG models on visual word counts gathered by SIFT++ with default parameters.
Table 3.17 shows the results for the different amount of cluster centers together with a RSM layer with 2028 hidden units. In comparison with Table 3.14, more RSM layer hidden units than cluster centers do perform slightly worse, but utilizing less hidden units seems to improve upon classification results.

Since for the 20 Newsgroups dataset the original RSM model uses very few (only 50) hidden units, we want to know, if this is the case here, too. While very few hidden units lead to less good results, the results of Table 3.17 and 3.18 suggest that employing half as many hidden units for the RSM layer than cluster centers is a good choice.

Experiment 3.5.4: Deep Neural Networks with RSM input layers: RSM->RBM->SIG models
Our initial goal was to use Deep Neural Networks with RSM input layers. However, the results on the 20 Newsgroups dataset already indicated that deep networks do not improve classification. The results of Table 3.19, where we add one more RBM layer in between the RSM input and the sigmoidal output layer, give evidence that this also holds for visual word counts (possibly even for all counts input data). Here, all RBM layers have 512 hidden units and are pre-trained for 100 epochs with learning rate 0.1.

Table 3.20 shows a comparison of the class Average Precision results for the conceptually different

[5]In particular, the best results are obtained with the following RSM->SIG models:
Lowe: Lowe SIFT implementation, K1024, RSM.H1024, RSM.E400, BP.E0
PCA-SIFT(36): Lowe SIFT implementation keypoints for PCA-SIFT(36), K2048, RSM.H2048, RSM.E400, BP.E0
T0_019E4: SIFT++ implementation with threshold and edge-threshold parameters set to 0.019 and 4.0, K512, RSM.H512, RSM.E1000, BP.E0
SIFT++: SIFT++ implementation with default parameters, K4096, RSM.H4096, RSM.E500,BP.E246
MBKM++: SIFT++ implementation with default parameters, K2048 (kmeans++ initialization), RSM.H2048, RSM.E250, BP.E0

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
SIG tf	13.79 (667)	14.76 (867)	16.17 (998)	15.78 (995)
NN tf-idf	11.23 (368)	18.57 (90)	24.41 (74)	24.75 (249)
NN tf	28.30 (29)	27.12 (105)	26.87 (489)	26.00 (484)
RSM.E100	31.18 (19)	31.66 (1)	31.34 (250)	30.55 (181)
RSM.E250	**32.33 (148)**	**33.08 (6)**	33.10 (250)	32.84 (51)
RSM.E400	31.90 (1)	32.94 (95)	**33.19 (0)**	32.96 (3)
RSM.E500	31.96 (250)	32.98 (8)	33.13 (0)	**33.28 (246)**
RSM.E750	31.53 (0)	32.45 (26)	32.64 (63)	32.21 (214)
RSM.E1000	31.55 (32)	31.80 (29)	32.16 (49)	31.80 (0)

Table 3.14: Mean Average Precision classification rates on visual word counts obtained by **Vedaldi's SIFT++** implementation with **default** parameters and Mini-batch k-Means.

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
SIG tf	13.55 (804)	15.26 (935)	15.03 (713)	15.87 (895)
NN tf-idf	11.66 (15)	19.42 (91)	23.49 (92)	24.00 (455)
NN tf	28.34 (23)	26.71 (22)	26.05 (500)	25.90 (499)
RSM.E100	31.34 (0)	30.85 (18)	31.16 (92)	30.43 (168)
RSM.E250	31.95 (249)	**32.63 (38)**	**33.22 (0)**	32.09 (250)
RSM.E400	32.05 (2)	32.39 (6)	32.83 (0)	**32.45 (249)**
RSM.E500	**32.24 (1)**	32.55 (6)	32.70 (115)	32.41 (128)
RSM.E750	32.01 (60)	32.36 (2)	31.83 (0)	31.34 (197)
RSM.E1000	31.61 (4)	32.12 (0)	31.47 (158)	30.94 (131)

Table 3.15: Mean Average Precision classification rates on visual word counts obtained by **Vedaldi's SIFT++ implementation with default parameters** and Mini-batch k-Means initialized with **kmeans++**.

	Lowe	PCA-SIFT(36)	T0_019E4	SIFT++	MBKM++
aeroplane	47.99	46.42	41.53	**51.58**	**51.58**
bicycle	32.24	35.02	22.21	**35.20**	32.38
bird	27.77	27.74	19.73	28.59	**30.93**
boat	37.40	31.88	22.73	**43.64**	43.27
bottle	**20.54**	18.03	10.87	19.56	13.74
bus	**25.08**	24.66	22.57	24.94	22.14
car	42.49	36.23	38.53	**43.79**	43.75
cat	34.87	31.69	28.48	33.89	**35.34**
chair	34.39	28.06	32.68	**36.35**	34.96
cow	**23.15**	17.05	22.67	20.47	21.35
diningtable	22.37	22.84	22.47	20.70	**24.24**
dog	31.00	31.73	29.69	**32.82**	31.94
horse	40.31	44.61	35.02	46.55	**46.90**
motorbike	26.96	28.56	21.29	29.31	**30.95**
person	69.93	68.74	66.05	**70.43**	70.38
potted_plant	13.36	17.97	**19.23**	14.67	15.81
sheep	24.58	23.69	19.34	25.54	**25.88**
sofa	**25.13**	23.15	24.73	22.48	24.31
train	36.67	37.55	33.17	35.50	**39.76**
tv/monitor	29.17	23.21	27.80	**29.66**	24.98
mean Average Precision	32.27	30.94	28.04	**33.28**	33.23

Table 3.16: Class Average Precisions resulting from different descriptors and kmeans++ initialization (MBKM) with RSM->SIG models, in particular best mAP performances from Tables 3.11 to 3.15.

\RSM.	K512, **H2048**	K1024, **H2048**	K2048, **H2048**	K4096, **H2048**
NN tf-idf	10.79 (41)	17.82 (32)	24.41 (74)	25.07 (293)
NN tf	27.86 (412)	26.82 (456)	26.87 (489)	25.04 (462)
RSM.E100	31.71 (16)	31.47 (8)	31.34 (250	30.60 (127)
RSM.E250	**32.85 (93)**	32.61 (0)	33.10 (250)	33.05 (244)
RSM.E400	32.65 (161)	**32.83 (140)**	**33.19 (0)**	**33.39 (0)**
RSM.E500	32.69 (231)	32.64 (214)	33.13 (0)	33.16 (0)
RSM.E750	32.63 (20)	32.12 (250)	32.64 (63)	32.96 (0)
RSM.E1000	32.45 (34)	32.12 (100)	32.16 (49)	32.02 (0)

Table 3.17: Mean Average Precision classification rates on visual word counts obtained by **Vedaldi's SIFT++ implementation with default parameters** and Mini-batch k-Means with **2048 hidden units**.

models so far plus the DualRSM model, introduced in Section 3.6.[6] The standard Feed-forward

[6]In particular, the best results are obtained using the following models:
RSM->SIG: SIFT++ with default parameters, K2048, RSM.H1024, RSM.E250, BP.E70

\RSM.	K2048, **H128**	K2048, **H256**	K2048, **H512**	K2048, **H1024**
NN tf-idf	24.47 (56)	24.62 (53)	24.51 (78)	23.63 (37)
NN tf	26.94 (43)	26.70 (57)	26.21 (238)	26.87 (275)
RSM.E100	23.15 (148)	27.54 (250)	29.77 (250)	30.98 (161)
RSM.E250	25.53 (1)	30.92 (12)	32.16 (77)	**33.65 (70)**
RSM.E400	27.28 (17)	31.83 (1)	32.47 (1)	33.43 (0)
RSM.E500	27.39 (11)	**32.63 (128)**	**32.48 (1)**	33.11 (6)
RSM.E750	28.44 (43)	32.48 (244)	32.23 (195)	32.75 (0)
RSM.E1000	**29.25 (97)**	32.19 (2)	32.17 (6)	32.16 (0)

Table 3.18: Mean Average Precision classification rates on visual word counts obtained by **Vedaldi's SIFT++ implementation with default parameters** and Mini-batch k-Means with **few hidden units**.

\RSM.	K512, H512	K1024, H1024	K2048, H2048	K4096, H4096
RSM.E100	31.54 (8)	31.15 (231)	29.90 (61)	24.99 (37)
RSM.E250	32.05 (42)	32.91 (117)	31.84 (243)	29.34 (197)
RSM.E400	31.46 (9)	32.72 (126)	**32.20 (143)**	**29.96 (182)**
RSM.E500	**32.27 (67)**	**33.35 (67)**	32.18 (249)	29.54 (239)
RSM.E750	31.99 (84)	32.20 (28)	31.85 (124)	28.81 (245)
RSM.E1000	31.99 (21)	32.58 (62)	30.77 (226)	28.20 (188)

Table 3.19: Mean Average Precision classification rates on visual word counts obtained by **Vedaldi's SIFT++ implementation with default parameters**, Mini-batch k-Means and Deep Belief Networks with RSM input layers. (RBM.H512, RBM.E100, RBM.L0.1 for all experiments) **(RSM->RBM->SIG)**.

Neural Networks perform better than a single sigmoidal layer as one would expect. The standard Neural Networks trained on tf-idf values perform less good than the ones trained on tf values. Nevertheless, Neural Networks with RSM input layers trained on tf values significantly outperform standard Feed-forward Neural Networks. Figures 3.7 and 3.8 display the recall precision curves from which the Average Precision scores are calculated.

Experiment 3.5.5: Support Vector Machines (SVM) on visual word counts and 128 element codes calculated by a Deep Auto-Encoder.

MBKM++: SIFT++ with default parameters, K2048 with kmeans++ initialization, RSM.H2048, RSM.E250, BP.E0
DBN: RSM->RBM->SIG model, SIFT++ with default parameters, K1024, RSM.H1024, RSM.E500, RBM.H512, BP.E67
SIG: sigmoidal layer only, SIFT++ with default parameters, K2048, BP.E998
NN tf: standard Feed-forward Neural Network fed with visual word counts (tf), SIFT++ with default parameters, K4096, 2048 hidden units, BP.E293
NN tf: standard Feed-forward Neural Network fed with tf-idf values, SIFT++ with default parameters, K512, 512 hidden units, BP.E29
DualRSM: left wing: SIFT++ with default parameters, K2048; right wing: all-to-all distance counts with 1000 bins; RSM.H2048, RSM.L0.0001, RSM.E1500

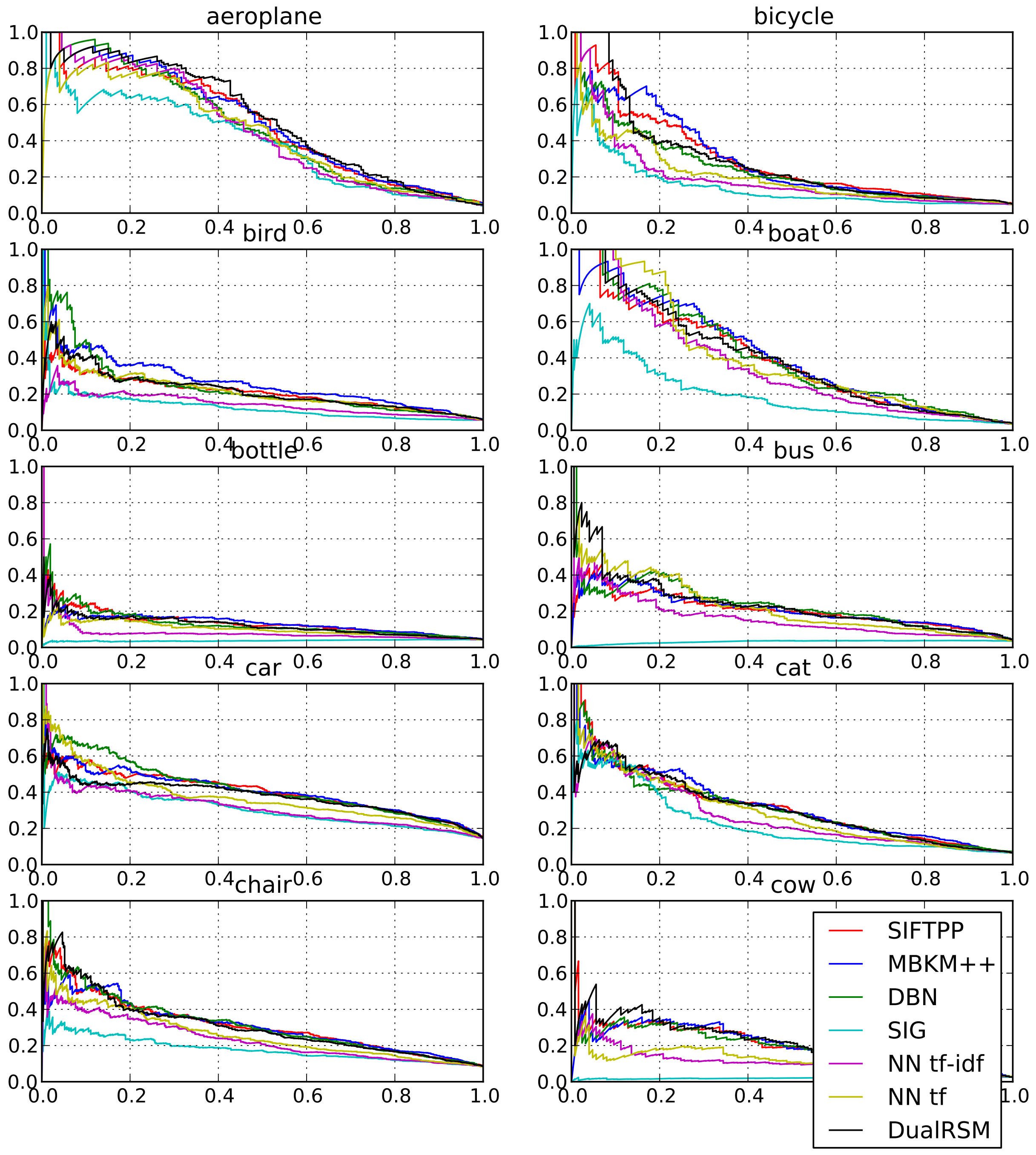

Figure 3.7: The corresponding recall precision curves to the values displayed in Table 3.16, classes *aeroplane* to *cow*. See in color for better visualization.

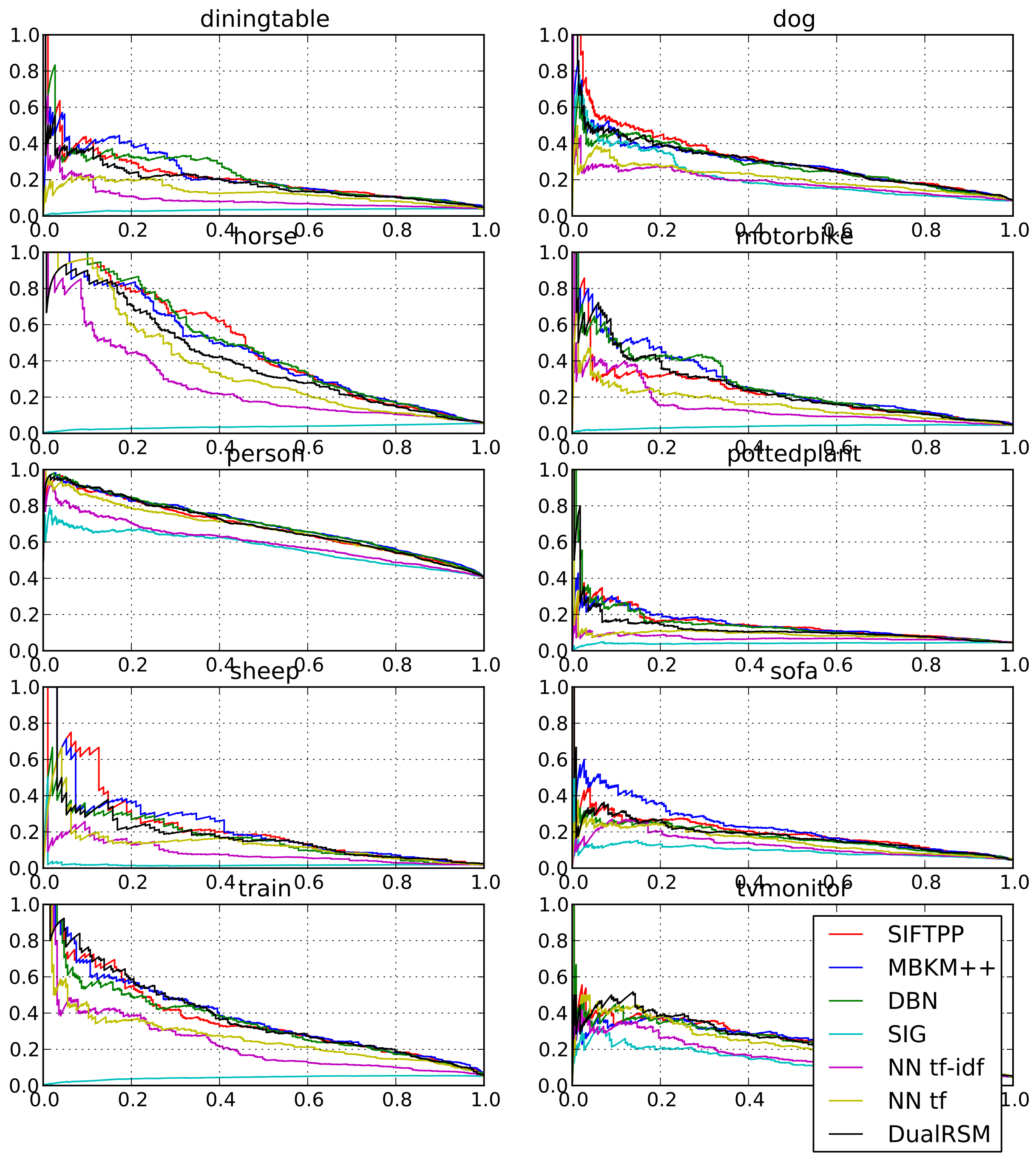

Figure 3.8: The corresponding recall precision curves to the values displayed in Table 3.16, classes *diningtable* to *tv/monitor*. See in color for better visualization.

	RSM->SIG	MBKM++	DBN	SIG	NN tf-idf	NN tf	DualRSM
aeroplane	50.12	51.58	48.29	40.99	46.79	46.24	**53.24**
bicycle	**33.06**	32.38	27.56	17.27	22.38	25.12	30.86
bird	26.73	**30.93**	27.38	18.89	15.56	24.78	23.80
boat	40.27	**43.27**	42.20	21.45	37.38	42.47	40.71
bottle	**20.57**	13.74	15.99	3.99	15.20	10.74	14.71
bus	21.62	22.14	**28.91**	3.50	21.80	27.16	27.48
car	43.95	43.75	**44.88**	34.03	35.80	39.80	39.01
cat	34.62	**35.34**	34.37	25.33	30.92	32.82	35.26
chair	34.67	**34.96**	34.73	18.11	26.28	30.62	34.27
cow	26.31	21.35	25.30	2.39	18.06	19.64	**26.91**
diningtable	26.21	24.24	**27.30**	3.70	16.91	13.37	24.67
dog	**34.92**	31.94	33.14	25.82	25.23	22.45	33.32
horse	48.56	46.90	**49.05**	5.02	29.57	38.48	42.59
motorbike	25.71	**30.95**	30.75	4.35	20.71	19.81	28.72
person	68.78	**70.38**	70.20	56.13	61.84	67.94	69.17
potted_plant	15.37	15.81	**20.10**	4.20	7.98	17.03	18.37
sheep	**26.95**	25.88	19.30	6.18	10.05	16.41	22.88
sofa	**25.31**	24.31	20.53	12.68	13.90	16.33	24.28
train	38.43	39.76	36.96	4.98	26.98	30.53	**40.03**
tv/monitor	**30.92**	24.98	30.16	14.49	18.24	24.32	27.12
mAP	**33.65**	33.22	33.35	16.17	25.07	28.30	32.87

Table 3.20: Class Average Precisions of the top-performers (mAP) of the conceptually different models: RSM->SIG on visual word counts obtained from SIFT++ descriptors with default parameters, plus with kmeans++ initialization (MBKM++), RSM->RBM->SIG (DBN), sigmoidal layer only (SIG), standard Feed-forward Neural Networks on visual word counts (NN tf) and tf-idf (NN tf-idf) values, and the DualRSM model.

We trained binary one-versus-all Support Vector Machines with linear kernels on both: directly on the visual word counts (tf), as well as on codes produced by a Deep Auto-Encoder with Replicated Softmax in- and output-layers. For this task we use Joachims [22] SVMlight[7].

The first SVM's are trained on visual word counts obtained via SIFT++ with default parameters and 2048 cluster centers calculated with MBKM on the train set. We optimize rather coarsely on the SVM hyperparameter C (trade-off between training error and margin), i.e. for each class we train a SVM with $C = 0.0001, 0.0003, 0.001, 0.01, 0.1$ and choose the one giving the best class Average Precision on the test set. The SVM's confidence in class prediction is rescaled to the interval $[0; 1]$. The value 0.0003 is the default parameter for C and calculated as $[\text{avg} x * x]^{-}1$ on the word counts from the train set.

For the Deep Auto-Encoder, the RSM layer is pre-trained for 500 epochs with 2048 hidden units

[7]`http://svmlight.joachims.org/`

	SVM on tf	SVM on DAE codes
aeroplane	4.72 (C=0.0001)	**35.86 (C=0.0001)**
bicycle	8.18 (C=0.003)	**16.49 (C=0.01)**
bird	7.18 (C=0.001)	**10.37 (C=0.0189)**
boat	4.18 (C=0.003)	**7.60 (C=0.1)**
bottle	4.68 (C=0.0001)	**6.33 (C=0.001)**
bus	4.20 (C=0.01)	**5.52 (C=0.1)**
car	16.20 (C=0.01)	**18.15 (C=0.001)**
cat	8.00 (C=0.0001)	**12.18 (C=0.001)**
chair	8.80 (C=0.1)	**12.51 (C=0.01)**
cow	3.50 (C=0.01)	**10.11 (C=0.0001)**
diningtable	12.60 (C=0.1)	**15.90 (C=0.0001)**
dog	**16.93 (C=0.0001)**	11.73 (C=0.01)
horse	6.09 (C=0.1)	**9.53 (C=0.0189)**
motorbike	5.37 (C=0.01)	**11.07 (C=0.01)**
person	46.01 (C=0.1)	**54.80 (C=0.01)**
pottedplant	5.59 (C=0.1)	**6.78 (C=0.0189)**
sheep	2.20 (C=0.003)	**12.10 (C=0.0189)**
sofa	5.38 (C=0.01)	**7.46 (C=0.01)**
train	**14.35 (C=0.01)**	8.19 (C=0.01)
tvmonitor	6.14 (C=0.01)	**10.72 (C=0.001)**
Mean Average Precision	9.51	**14.17**

Table 3.21: Class Average Precision results on the test set for one-versus-all Support Vector Machines (SVM) on visual word counts and codes (vectors of size 128) calculated by a Deep Auto-Encoder (DAE) on visual word counts (optimal hyperparameter C in brackets.

and learning rate 0.0001 on the same word counts as the first SVM's. There are two subsequent layers, pre-trained as standard RBMs for 100 epochs with learning rate 0.1. The encoder part of the Deep Auto-Encoder model can be characterized as follows:
RSM.H2048->RBM.H1024->RBM.H128. SVM's are then trained on the resulting 128 element codes carried out on the train set with hyperparameters $C = 0.0001, 0.001, 0.01, 0.0189, 0.1$ as described above. Here, 0.0189 is the calculated default parameter for C.

The results presented in Table 3.21 are unexpectedly bad, but strangely, for some classes the Average Precision is good. Also, the SVM trained on the codes received by the Deep Auto-Encoder are clearly better than for the SVM directly trained on word counts. This might me due to the fact that SVMlight targets on sparse input data.

Experiment 3.5.6: Linear embedding using an RSM model with 2 and 3 hidden units.
Here, we try to perform a linear embedding comparable e.g. to t-SNE by Van Der Maaten and Hinton [55], which was already successfully achieved using EFH models, e.g. with UP-LSI

from Welling et al. [58]. Therefore, we used a RSM model with only 2 hidden units, trained for few epochs (5 to 100) and very small learning rates. Afterwards, the hidden units' activations are visualized in a scatter plot. There was no clear clustering recognizable, even not if we restrict training and visualization to images with exactly one object of two different classes. We also tried to train a Deep Auto-Encoder producing 2 element codes and used the resulting activations on the code layer for a linear embedding with comparable results. Training a Deep Auto-Encoder that produces 128 element codes and applying t-SNE on these codes approaches a linear embedding, but only for some combinations of few object classes (images containing only pictures from one of the two or three classes).

In summary, two-layer Neural Networks with Replicated Softmax input layers outperform two-layer standard Feed-forward Neural Networks for about 5% in mean Average Precision at the task of image object classification, where about 1% probably stems from overfitting of the RSM layer. The kmeans++ initialization of Mini-batch k-Means does not improve upon classification results, but may lead to slightly more stable results, despite it's expensiveness with regards to computation time. We can not achieve as good results as e.g. Zhou et al. [66], but we come close to the results of Bengio et al. [4]. In contrast to both, we do not collect SIFT descriptors on a fixed, usually very fine grid on the images, but use standard SIFT detector implementations on the images. The more SIFT descriptors are obtained from the images, the better the results become. Removing large fractions of information from the descriptors (PCA-SIFT(36)) does lower classification results. Results from reducing visual word counts to 128 element codes using a Deep Auto-Encoder and training binary one-versus-all Support Vector Machines on the codes perform strangely bad. We were not able to achieve a meaningful linear embedding using the Replicated Softmax model. We also tried it once with three hidden units, looking slightly better, but far from meaningful.

3.6 DualRSM model & visual word all-to-all distance counts

We experimented with an idea to extend the Replicated Softmax input layer of our pipeline to incorporate spatial information about the keypoints, since it is abandoned by the quantization of visual words into word counts. In 2007, Burghouts et al. [10] showed that all-to-all distances measured with L_p-norm similarity measures on feature vectors obtained from images are Weibull distributed, 'if the feature vectors are correlated and non-identically distributed'. Therefore, we added a second wing to the Replicated Softmax. This is an application of Dual Wing Harmoniums introduced by Xing et al. [62]. We call this the **DualRSM** model. We feed the DualRSM model with visual word counts and all-to-all distance counts. This can also be seen as a variant of what Zhang et al. [65] call term connection frequency for images rather than text documents.

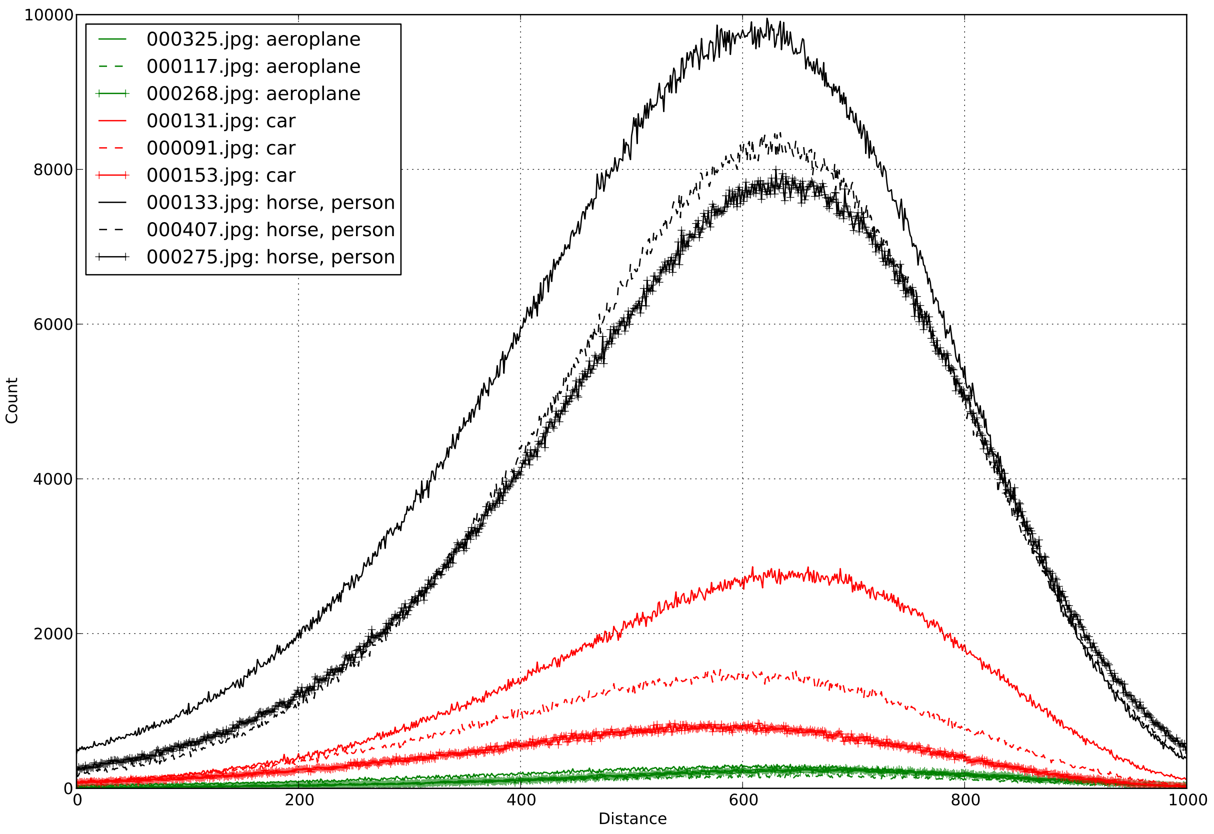

Figure 3.9: All-to-all distances distributions of visual words from selected training images and three different classes. The distance counts are Weibull distributed as evinced by Burghouts et al. [10]. See in color for better visualization.

First, the all-to-all distances[8] of the feature vectors, obtained by SIFT++ with default parameters from the PASCAL VOC 2007 images, appear to obey Weibull distributions, as demonstrated

[8]one distance for each pair of all feature vectors from one image vectors

in Figure 3.9. The Weibull distribution can be defined as follows (without location parameter):

$$f(x, \lambda, k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} \exp\left(-\left(\frac{x}{\lambda}\right)^{k}\right), \quad x, \lambda, k \geq 0 \tag{3.6}$$

where k is the shape and λ is the scale parameter. Figure 3.9 visualizes the idea that the all-to-all distances of the feature vectors obey a different Weibull distribution for each class (however, here we selected images that empower this hypothesis), i.e. the all-to-all distances of the keypoints from one image hold discriminative information about the objects present in it. The all-to-all distance counts are, likewise to the respective SIFT keypoints, scale- and rotation-invariant.

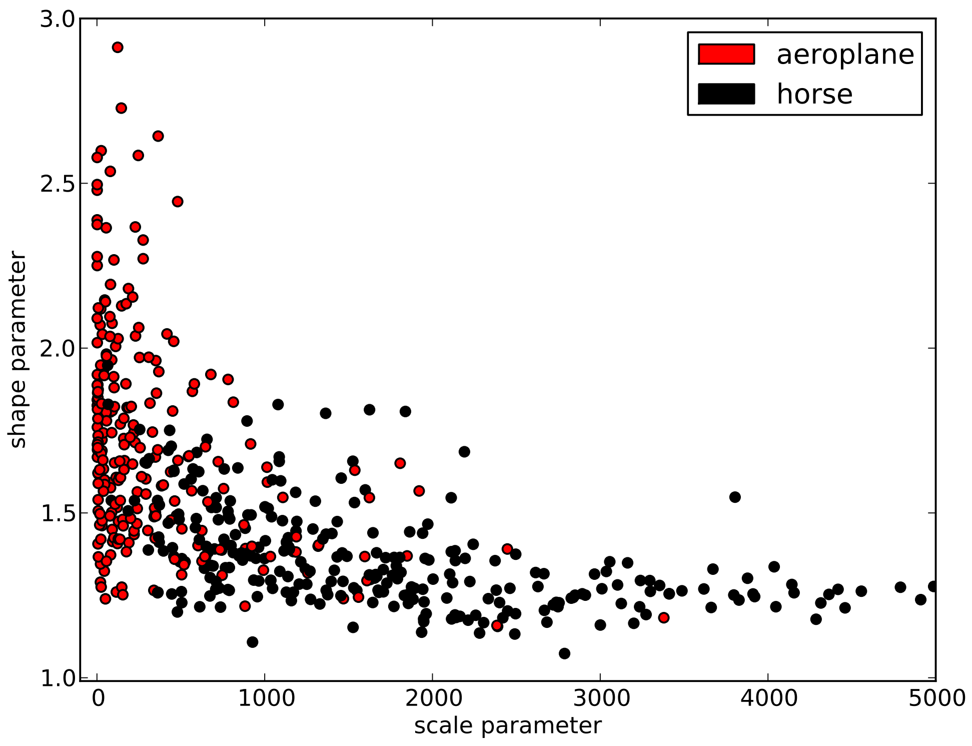

Figure 3.10: Estimates of the assumed Weibull distribution's shape and scale parameters for each training set image containing *at least one* object from one of the two classes *aeroplane* or *horse*. See in color for better visualization.

Figure 3.10 reinforces that the Weibull parameters estimates are different for certain different classes. Figures 3.11 and 3.12 depict the estimates of the shape and scale parameters of the assumed Weibull distributions of all images containing *at least one* object from the respective class in the training set and the test set, respectively. Obviously, several classes share similar parameter characteristics. The scale parameter directly relates to the number of SIFT keypoints detected in an image. The images picturing an object of the *aeroplane* class, appear to have very few descriptors (this might e.g. be due to the 'empty' heaven background often found in such images). Using a standard detector, larger images result in a larger amount of descriptors and

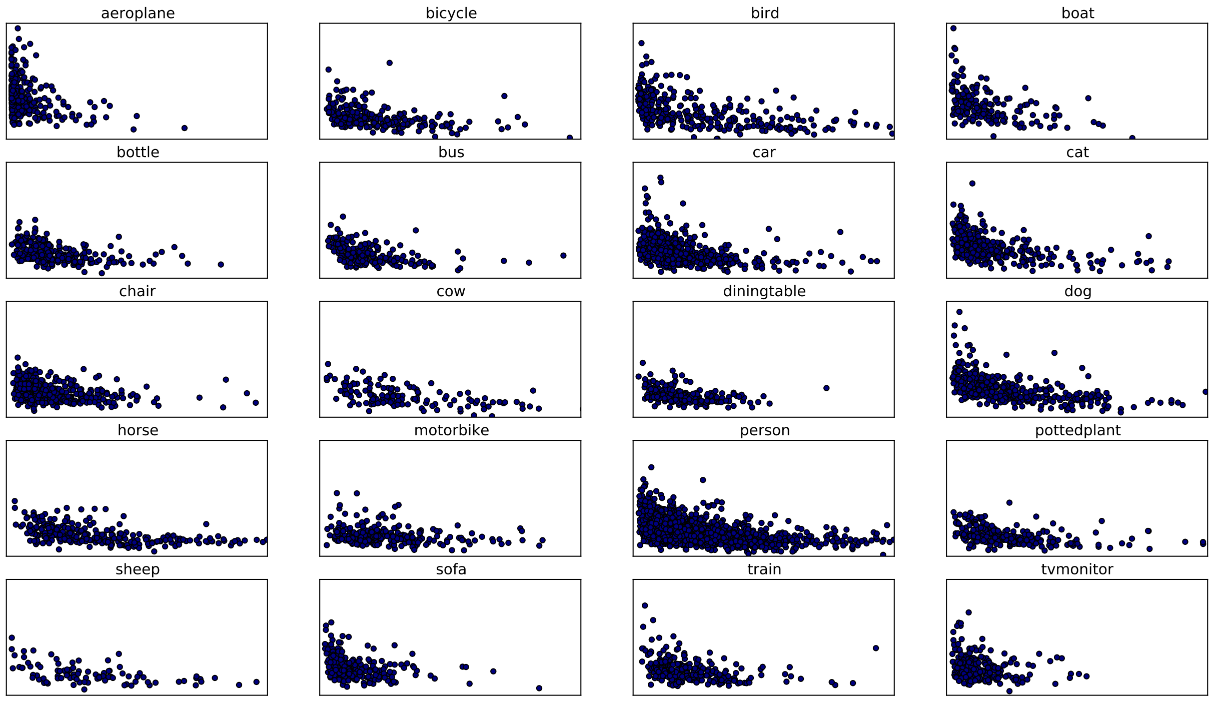

Figure 3.11: Shape and scale parameter estimates (**training set**), for details see text.

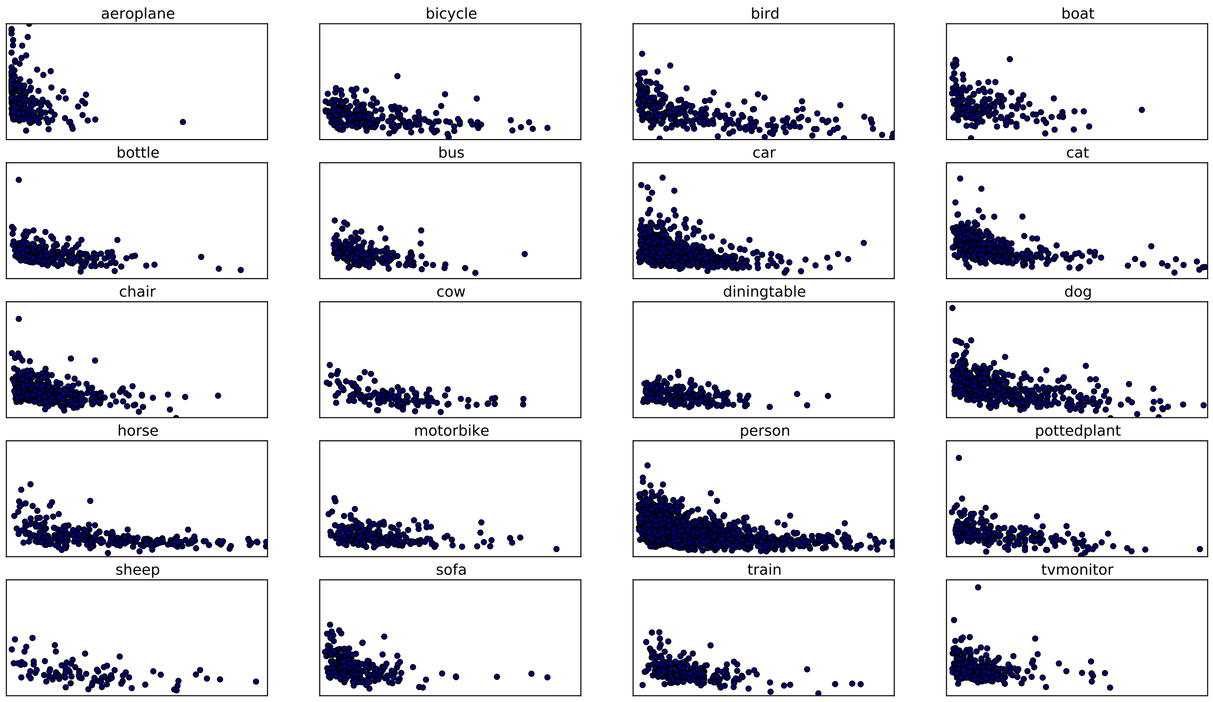

Figure 3.12: Shape and scale parameter estimates (**test set**), for details see text.

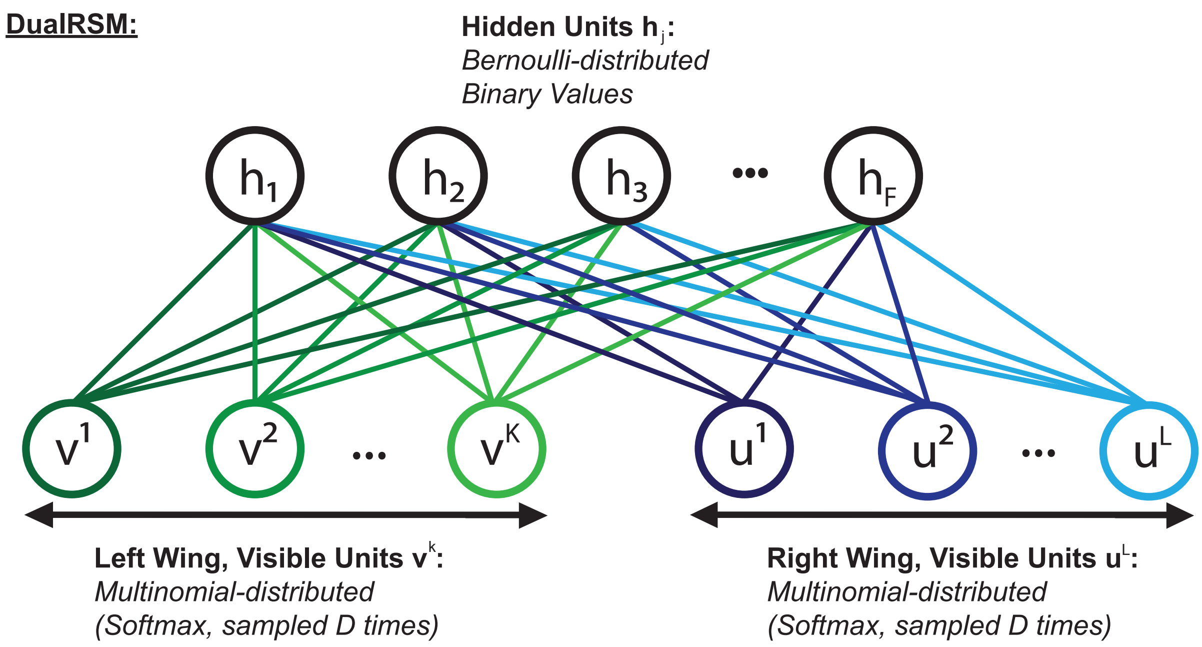

Figure 3.13: Architecture of the DualRSM model. Two wings on the visible random variables layer allow for two different types of input data (input data in histogram form, e.g. term frequencies and all-to-all distance counts).

thus images of different sizes are problematic. However, one could remedy this problem by using the common approach of partitioning the image into blocks, see e.g. Bengio et al. [66]. However, Schroeder [50] reports that Weibull parameter estimation algorithms are often unstable. We restricted the range of relevant distances between SIFT descriptors to the interval $[0.7, 1.3]$ (due to this, we do not need the location parameter of the Weibull distribution). This range is discretized into 1000 bins in order to obtain a counts histogram. Figure 3.13 visualizes the architecture of the DualRSM model extended by adding a second wing on the visible units layer. With all parameters adopted from the RSM model described in Subsection 2.7.7 the energy function is extended to:

$$E(\hat{V}, \hat{U}, h) = -\sum_{k=1}^{K} h^{\mathrm{T}} W^k \hat{V}^k - \sum_{l=1}^{L} h^{\mathrm{T}} \Psi^l \hat{U}^k - {a^k}^{\mathrm{T}} \hat{V}^k - {c^l}^{\mathrm{T}} \hat{U}^l - D h b^{\mathrm{T}} \tag{3.7}$$

where weight matrix Ψ, input count data matrix (or histogram matrix) $\hat{U}$ and bias vector c belong to the second wing. The sampling in this model is easily carried out as follows:

$$p(\hat{V}_i^k = 1|h) = \frac{\exp(a_i^k + \sum_{j=1}^{F} h_j W_j^k)}{\sum_{q=1}^{K} \exp(a_i^q + \sum_{j=1}^{F} h_j W_j^q)} \tag{3.8}$$

$$p(\hat{U}_i^l = 1|h) = \frac{\exp(c_i^l + \sum_{j=1}^F h_j \Psi_j^l)}{\sum_{p=1}^L \exp(c_i^p + \sum_{j=1}^F h_j \Psi_j^p)} \tag{3.9}$$

$$p(h_j = 1|\hat{V}, \hat{U}) = \sigma\left(Db_j + \sum_{k=1}^K W_j^k \hat{V}^k + \sum_{l=1}^L \Psi_j^l \hat{U}^l\right) \tag{3.10}$$

The extension to the second wing is a direct application of Dual Wing Harmoniums presented by Xing et al. [62], plus weight sharing in both wings. Since we have two types of input data for each record, the scaling of the hidden biases remains unchanged. While this gives a theoretical justification, the implementation of the RSM model does not need to be changed here, the two input data matrices are simply concatenated. Due to a large discrepancy of maximum values, we rescale the distance counts by normalizing it to the maximum value of the visual word counts and rounded it to the nearest integer.

RSM.H, RSM.L	H3072, L0.0001	H3072, L0.00005	H3072, L0.00001	H2048, L0.0001
RSM.E100	26.81	23.87	18.64	27.22
RSM.E250	29.76	29.06	21.02	29.10
RSM.E400	31.49	30.89	24.18	31.20
RSM.E500	31.28	30.43	24.20	31.72
RSM.E750	30.93	31.74	26.41	32.82
RSM.E1000	31.35	**32.74**	26.84	32.41
RSM.E1250	**31.62**	32.53	**28.36**	32.56
RSM.E1500	31.33	32.36	26.53	**32.86**
RSM.E1750	31.54	**32.74**	26.94	31.25
RSM.E2000	31.55	31.21	29.37	32.24

Table 3.22: Mean Average Precision rates obtained by the DualRSM model, where one wing is fed with visual word counts obtained by SIFT++ with default parameters ($K = 2048$) and the other wing is fed with all-to-all distance counts ($B = 1000$ bins).

We trained the DualRSM model as part of a DualRSM->SIG model with different learning rates, constant momentum rate of 0.9, where the SIG layer is trained for 30 epochs with learning rate 0.1, but without error backpropagation. The DualRSM weights are initialized as described for the RSM model in Section 3.1. First experiments suggested that the second wing improves upon classification. Unfortunately, we were not able to validate these results in the long run and adding the second wing fed with all-to-all distance counts turns out to decrease mean Average Precision results in comparison to the respective RSM->SIG model trained on word counts only, see Table 3.22. The error backpropagation did not work and therefore is neglected, i.e. the mAP results come from pre-training the DualRSM and the sigmoidal layer solely.

In summary, the DualRSM model trained on the combination of visual word counts and their respective all-to-all distance counts does not improve upon image object classification. The second wing of the DualRSM model could, also be fed with other input data, e.g. in an image retrieval engine that combines images and keywords as input from the user.

4 Conclusion

The primary goal of this thesis was to realize a bag-of-visual-words framework for image retrieval that uses the Replicated Softmax model as part of a Deep Belief Network on visual word counts obtained by vector quantization of SIFT descriptors extracted by a standard detector rather than keypoints extracted at all points of a fine grid over the images.

First, Sculley's Mini-batch k-Means algorithm appears to be an attractive alternative to the Hierarchical and Approximate variants of k-Means for the quantization of visual words due to its time- and memory efficiency, as well as its easy parallelization on GPUs utilizing Gnumpy. A direct comparison would be very useful and remains an effort for the future. While loading all descriptors into main memory is a problem, subsampling of descriptor space appears to be a working solution. We used 1 million subsamples, but this could be explored further.

Secondly, the Replicated Softmax model turns out to be a powerful tool in both use cases: as a standalone model or as part of a Neural Network. For both applications the most important hyperparameter is the number of epochs the Replicated Softmax model is trained. However, the Replicated Softmax model is prone to overtraining and needs to be trained for several hundred epochs (2 to 40 hours). This is probably due to the hidden biases scaling, requiring the learning rate to be relatively small. Hence, training the Replicated Softmax is a clear bottleneck in our pipeline. Thus, one obvious goal for the future is to develop parallelized implementation of the Replicated Softmax model. We were not able to parallelize it with Gnumpy due to the required sampling from multinomial random variables. Eventually though, using the logsum-trick in the implementation might help to use larger learning rates.

Our initial goal was to employ Deep Neural Networks. However, for document as well as image object classification, networks with more than two layers did not improve upon classification performance in our pipeline. Eventually, this might hold for all counts input data. Despite this, two-layer Neural Networks with Replicated Softmax input layers perform about 5% better than standard Feed-forward Neural Networks on the image object classification task, which is a very good result.
Neural Networks with Replicated Softmax input layers also perform very well for document classification on the 20 Newsgroups dataset with absolute results comparable to LDA and DiscLDA on dictionaries based on most frequent word counts and highest information gain, respectively. The latter with classification rates greater than 80%. We can reproduce the retrieval results from Salakhutdinov and Hinton on the 20 Newsgroups dataset using our standalone Replicated Softmax implementation. By using a dictionary based on highest information gain rather than on most frequent word counts, we significantly exceed the results from the original Replicated Softmax paper for document retrieval on the 20 Newsgroups dataset.
In both applications, the absolute classification performance heavily depends upon the input

data, e.g. simply by selecting a dictionary by information gain rather than most frequent occurrences can result in a gain of ca. 5% absolute classification performance. Likewise, the results of the image object classification task tend to become better, the more descriptors are used. Due to this fact, others typically extract descriptors along a very fine grid resulting in a huge amount of descriptors. If information is cut out, the results suffer, too.[1] So, two interesting questions would be how good our pipeline performs on descriptors obtained, on the one hand along grid-points, or on the other hand by a standard detector and reduced in number by selecting distinctive descriptors only, as described by Kang et al. [23].

We trained binary one-versus-all Support Vector Machines on both: the visual word counts (tf) and 128-element codes obtained by a Deep Auto-Encoder trained on the visual word counts. The absolute classification results were quite bad, especially since leading papers achieve very good results using Support Vector Machines. However, we did not spent much time on this. Nevertheless, the Support Vector Machines trained on the codes obtained by the Deep Auto-Encoder are significantly better.

Welling et al. [58] succeeded in carrying out a linear embedding using the UP-LSI model. We experimented with Replicated Softmax models with two or three hidden units trained with very small learning rates for few epochs, but no clearly visible clustering occurred. An open task for the future is the application of unit-variance Gaussian hidden units in the RSM model rather than Bernoulli latent variables[2].

The DualRSM model as input layer of a Neural Network was an appealing idea to me. In the end, when fed with visual word counts and their respective all-to-all distance counts the results were slightly worse than with a comparable single RSM model input layer. It could improve, at least a bit, the classification results on the class of *aeroplane* objects. This is due to the coupling of the scale parameter of the assumed Weibull distributions to number of descriptors. The images containing at least one object of type *aeroplane* typically have much less descriptors than images where objects from the other classes are present, which is eventually due to the fact that aeroplanes are often photographed in the sky. A conceivable way of improvement would be to partition the image and to use distance counts from within the partitions. Moreover, one could also use the DualRSM model to combine two different word count inputs, visual word counts plus a dictionary stemming from annotation data, thereby integrating input data obtained by unsupervised learning and manually added meta data.

Modeling spatial relationships and being able to incorporate such information would be of great advantage. A difficult, but eventually promising idea could be to shift from image to object level on the input side of the pipeline. An idea for this would be for example to automate image segmentation first in a way that optimizes for large but independent connected components and to extract keypoints from within these connected components and subsequently fed into the bag-of-visual-words pipeline.

[1]In this work we merely experimented with the implementation given by the authors of the criticized PCA-SIFT paper.
[2]The mathematics work out smoothly, but we did not implement it.

A Development environment

All experiments were carried out on a system with the following configuration:

Hardware:

- **Processor:** Intel Core i5-2500, 4 x 3300 MHz, L1-Cache: 4 x 256 KByte , L2-Cache: 6144 Kbyte
- **Main Memory:** G.Skill DIMM 8GB DDR3-1333
- **Graphics:** ASUS ENGTX580 DCII/2DIS, 1536MB, 384 Bit, PCIe 2.0 x16, with NVIDIA GeForce GTX580 (512 CUDA Cores)
- **Motherboard:** ASUS P8P67 Deluxe R.3.0
- **Hard Disk Drive:** Hitachi HDS722020ALA330 2TB, SATA300, 8.2ms, 32MB Cache, 7200 RPM

Software:

- **Operating System:** Ubuntu Linux 10.10 64-bit, Kernel 2.6.35
- Python 2.6.6, Numpy[1] 1.5.1 with self-build ATLAS 8.3 (for Netlib's LAPACK 3.3.0)
- SciPy[2] 0.9.0
- scikits.learn[3] 0.8
- matplotlib 1.0.0
- CUDA Toolkit[4] 3.0 64-bit for Ubuntu 9.04 (v. 3.0 for Cudamat compatibility)
- Cudamat[5] 0.3 by Mnih [36] (limited to 32-bit floats)
- Gnumpy[6] by Tieleman [54] (limited to 32-bit floats)
- SIFT++ 0.8.1[7] by Vedaldi [56]
- VLFeat[8] 0.9.9 by Vedaldi [56]

[1] `http://numpy.scipy.org/`, `http://math-atlas.sourceforge.net/`, `http://www.netlib.org/lapack/`
[2] `http://www.scipy.org/`
[3] `http://scikit-learn.sourceforge.net/`
[4] `http://developer.nvidia.com/cuda-toolkit-30-downloads`
[5] `http://code.google.com/p/cudamat`
[6] `http://www.cs.toronto.edu/~tijmen/gnumpy.html`
[7] `http://www.vlfeat.org/~vedaldi/code/siftpp.html`
[8] `http://www.vlfeat.org/`

B Batch-, online- and minibatch-methods & momentum

Iterative methods are widely used in various scientific disciplines, especially if exact methods are computationally intractable. For our purposes assume a dataset $D = \{d_0, \dots, d_{N-1}\}$ with N records and a model with parameters θ. Starting with an initialization that is easy to calculate, an iterative procedure approaches a solution by updating the model parameters succesively. In each step the model parameters are updated to find a 'better' approximation θ^{t+1}. The update function f depends upon the dataset and the current parameter values at step t (and target values in a supervised setting):

$$\theta^{t+1} := \theta^t + \eta * f(D, \theta^t) \tag{B.1}$$

A very common iterative method is **gradient descent** that updates the model by taking steps along the negative gradient (the gradient is a vector pointing towards the direction of the greatest ascent) of some objective function $E(D, \theta)$:

$$\theta^{t+1} := \theta^t - \eta * \nabla E(D, \theta^t) \tag{B.2}$$

If the update function or in particular the gradient descent function is of the following form:

$$\nabla E(D, \theta^t) = \sum_{n=1}^{N} e(d_n, \theta^t) \tag{B.3}$$

where $e(d_n, \theta^t)$ is the gradient function expressed in terms of a single dataset record d_n, then an **online method**, also called **stochastic gradient descent (SGD)** variant can be applied:

$$\theta^{t+1} := \theta^t - \eta * e(d_n, \theta^t) \tag{B.4}$$

The advantage of online methods is that each record of the dataset can be presented once at a time in contrast to running through the whole dataset for each parameter update, what is called a **batch method**. However, online methods also experience a drawback, since running over each record individually is computationally usually inefficient and often comprises high variance. A trade-off solution to both problems can be the **minibatch** variant of SGD.

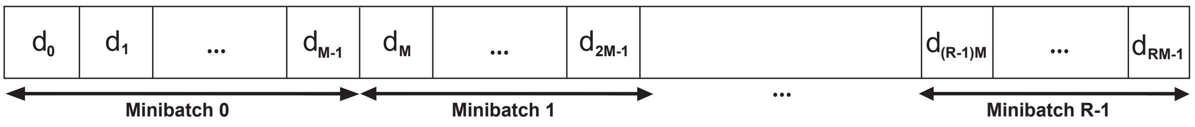

Figure B.1: Partitioning of a dataset $D = (d_0, \ldots, d_N)$ into R chunks, so-called minibatches, of equal size M.

Assuming the dataset can be partitioned into R chunks of equal size M:
$D = \{d_1, \ldots, d_{1*M}, d_{1*M+1}, \ldots, d_{2M}, \ldots\ldots, d_{RM}, \ldots, d_N\}$, cf. Figure B.1 then a parameter update r takes the form:

$$\theta^{t+1} := \theta^t - \eta * \left(\sum_{n=r*M}^{r*M+M-1} e(d_n, \theta^t) \right) \tag{B.5}$$

Often the calculation of M updates at once is computationally more efficient by exploiting linear algebra package subroutines. In order to carry out true SGD, in each iteration over a minibatch the records of a minibatch would have to be chosen randomly from the complete set of records. However, this would cause a lot overhead and running over fixed minibatches over the course of training often works well, too. Note that varying the minibatch size does affect the parameter updates and usually leads to different solutions despite equivalent initializing, cf. Hinton [16]. Hence, choosing the minibatch size is a crucial task. Hinton suggests to choose the minibatches such that a record from each class is contained.

Another common option to stabilize gradient descent methods is by using a **momentum** term, see. Hinton [16]. It delays the full effect of the current parameter update $\delta\theta^t$ as follows:

$$\delta\theta^t := -\eta * f(D, \theta^t) + \alpha\delta\theta^{t-1} \tag{B.6}$$

with momentum rate α. Using Hinton's ball analogy, it forces the ball rolling downhill (gradient descent) to react less nervous to current direction changes and thus stabilizes learning.

C Mathematical derivations

C.1 Sigmoid, softmax and hyperbolic tangent function derivatives

Tangens hyperbolicus (Eq. 2.31) derivative:
The derivative of the tangens hyperbolicus function can be expressed in terms of itself:

$$\frac{\partial \tanh(a)}{\partial a} = \frac{\partial}{\partial a}\frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{(\exp(a) + \exp(-a))^2 - (\exp(a) - \exp(-a))^2}{(\exp(a) + \exp(-a))^2} \tag{C.1}$$

$$= 1 - \tanh^2(a) \tag{C.2}$$

Logistic sigmoid function (Eq. 2.12) derivative:
Likewise, the derivative of the logistic sigmoid function can be expressed in terms of itself:

$$\frac{\partial \sigma(a)}{\partial a} = \frac{\partial}{\partial a}\frac{1}{1 + \exp(-a)} = \frac{\partial}{\partial a} = (1 + \exp(-a))^{-1} \tag{C.3}$$

$$= \frac{-1}{1 + \exp(-a)^2}(-\exp(-a)) = \frac{\exp(-a)}{(1 + \exp(-a))^2} \tag{C.4}$$

$$= \sigma(a)\left(\frac{\exp(-a)}{1 + \exp(-a)}\right) = \sigma(a)\left(\frac{1 + \exp(-a)}{1 + \exp(-a)} - \frac{1}{1 + \exp(-a)}\right) \tag{C.5}$$

$$= \sigma(a)(1 - \sigma(a)) \tag{C.6}$$

Softmax function (Eq. 2.17) derivative:
In order to calculate the derivative of the softmax function, a case study for $j = k$ and $j \neq k$ is necessary. For $j = k$ it follows that:

$$\frac{\partial y_k}{\partial a_k} = \frac{\partial}{\partial a_k}\frac{\exp(a_k)}{\sum_j \exp(a_j)} = \frac{\partial}{\partial a_k}\exp(a_k)\left(\sum_j a_j\right)^{-1} \tag{C.7}$$

$$= \exp(a_k)\left(\sum_k \exp(a_j)\right)^{-1} + \exp(a_k)\frac{\partial}{\partial a_k}\left(\exp(a_k) + \sum_{j\neq k}\exp(a_j)\right)^{-1} \tag{C.8}$$

$$= \frac{\exp(a_k)}{\sum_j \exp(a_j)} - \exp(a_k)\left(\exp(a_k) + \sum_{j\neq k}\exp(a_j)\right)^{-2}\exp(a_k) \tag{C.9}$$

$$= \frac{\exp(a_k)}{\sum_j \exp(a_j)} - \frac{(\exp(a_k))^2}{\left(\sum_j \exp(a_k)\right)^2} = y_k - y_k^2 \tag{C.10}$$

$$= y_k(1 - y_k) \tag{C.11}$$

While for $j \neq k$ the following holds:

$$\frac{\partial y_k}{\partial a_j} = \frac{\partial}{\partial a_j}\frac{\exp(a_k)}{\sum_j \exp(a_j)} = \frac{\partial}{\partial a_j}\exp(a_k)\left(\sum_j a_j\right)^{-1} \tag{C.12}$$

$$= 0 + \exp(a_k)\frac{\partial}{\partial a_j}\left(\sum_j \exp(a_j)\right)^{-1} = -\exp(a_k)\left(\sum_j a_j\right)^{-2}\exp(a_j) \tag{C.13}$$

$$= -\frac{\exp(a_j)\exp(a_k)}{\left(\sum_j a_j\right)^{-2}} \tag{C.14}$$

$$= -y_k y_j = y_k(0 - y_j) \tag{C.15}$$

These two partial results can be conveniently combined using the elements of the identiy matrix I_{kj}:

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \tag{C.16}$$

C.2 Cross-entropy error function gradients

The gradient of the **Logistic Regression** cross-entropy error function is calculated as follows. Given Eq. 2.14 one starts with calculating $\frac{\partial E(w)}{\partial y_n}$:

$$\frac{\partial E(w)}{\partial y_n} = \frac{\partial}{\partial y_n} - \sum_{n=1}^{N}(t_n \ln y_n + (1-t_n)\ln(1-y_n)) \tag{C.17}$$

$$= -\frac{\partial}{\partial y_n}(t_n \ln y_n + (1-t_n)\ln(1-y_n)(-1)) \tag{C.18}$$

$$= -\left(\frac{1}{t_n} - \frac{1-t_n}{1-y_n}\right) = \frac{y_n(1-t_n) - t_n(1-y_n)}{y_n(1-y_n)} \tag{C.19}$$

$$= \frac{y_n - y_n t_n - t_n + y_n t_n}{y_n(1-y_n)} \tag{C.20}$$

$$= \frac{y_n - t_n}{y_n(1-y_n)} \tag{C.21}$$

Next, $\frac{\partial y_n}{\partial a_n}$ with $y_n := \sigma(a_n)$ is calculated using the result of C.1:

$$\frac{\partial y_n}{\partial a_n} = \frac{\partial}{\partial a_n}\sigma(a_n) = \sigma(a_n)(1-\sigma(a_n)) = y_n(1-y_n) \tag{C.22}$$

Furthermore, $\nabla_w a_n$ amounts to:

$$\nabla_w a_n = \nabla_w w^{\mathrm{T}}\phi_n = \phi_n \tag{C.23}$$

Using the chain rule on $\frac{\partial E(w)}{\partial w}$ and the results of Equations C.21, C.22 and C.23, the derivative of Eq. 2.14 w.r.t. w amounts to:

$$\frac{\partial E(w)}{\partial w} = \sum_{n=1}^{N} \frac{\partial E(w)}{\partial y_n} \frac{\partial y_n}{\partial a_n} \nabla_w a_n = \sum_{n=1}^{N} \frac{y_n - t_n}{y_n(1-t_n)} y_n(1-t_n)\phi_n \tag{C.24}$$

$$= \sum_{n=1}^{N} (y_n - t_n)\phi_n \tag{C.25}$$

Note that the leading minus sign of Eq. 2.14 is already taken into account in Eq. C.17.

The gradient of the **Multiclass Logistic Regression** cross-entropy error function is calculated similarly. Given Eq. 2.20 one starts again with calculating $\frac{\partial E(w)}{\partial y_{nk}}$:

$$\frac{\partial E(w_1, \ldots, w_K)}{\partial y_{nk}} = \frac{\partial}{\partial y_{nk}} - \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk} = -\frac{\partial}{\partial y_{nk}} t_{nk} \ln y_{nk} \tag{C.26}$$

$$= -\frac{t_{nk}}{y_{nk}} \tag{C.27}$$

And $\nabla_{w_j} a_j$ amounts to:

$$\nabla_{w_j} a_j = \nabla_{w_j} w_j^{\mathrm{T}} \phi_n = \phi_n \tag{C.28}$$

Using analogously the chain rule on $\frac{\partial E(w)}{\partial w_j}$ and the results of Equations C.27, C.16 and C.28, the derivative of 2.20 w.r.t. w_j amounts to:

$$\frac{\partial E(w_1, \ldots, w_K)}{\partial w_j} = \sum_{n=1}^{N}\sum_{k=1}^{K} \frac{\partial E(w)}{y_{nk}} \frac{y_{nk}}{a_j} \nabla_{w_j} a_j = \sum_{n=1}^{N}\sum_{k=1}^{K} -\frac{t_{nk}}{y_{nk}} y_{nk} \left(I_{nk} - y_{nj}\right) \phi_n \tag{C.29}$$

$$= \sum_{n=1}^{N} (y_{nk} - t_{nj})\phi_n \tag{C.30}$$

Note that the leading minus sign of Eq. 2.20 is already taken into account in Eq. C.26. The sum over k vanishes in the last step, since due to the 1-of-K coding scheme $\forall n \in \{1, \ldots, N\}$: $\sum_{k=1}^{K} t_{nk} = 1; t_{nk} \in \{0, 1\}$ holds, i.e. only one addend of the sum for each n remains.

C.3 Conditional independence and sampling in the RBM model

Given the following energy function:

$$E(v, h) = -\sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - \sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{M} b_i h_i = -v^{\mathrm{T}} W h - a^{\mathrm{T}} v - b^{\mathrm{T}} h \tag{C.31}$$

with visible units vector $v = (v_1, \ldots, v_D), v_i \in \{0, 1\}$, hidden units vector $h = (h_1, \ldots, h_F), h_j \in \{0, 1\}$, bias vector for the visible units $a = (a_1, \ldots, a_D), a_i \in \mathbb{R}$, bias vector for the hidden units

$b = (b_1, \ldots, b_F)\,, b_i \in \mathbb{R}$ and $W_{ij} \in \mathbb{R}$ being the coupling matrix (weight matrix), the conditional distribution $p(v|h)$ is calculated following Freund and Haussler [14] as follows:

$$\begin{aligned}
p(v|h) &= \frac{\exp\left(-E(v,h)\right)}{\sum_{\hat{v}} \exp\left(-E(\hat{v},h)\right)} && \text{(C.32)}\\
&= \frac{\exp\left(v^{\mathrm{T}}Wh + a^{\mathrm{T}}v + b^{\mathrm{T}}h\right)}{\sum_{\hat{v}} \exp\left(\hat{v}^{\mathrm{T}}Wh + a^{\mathrm{T}}\hat{v} + b^{\mathrm{T}}h\right)} && \text{(C.33)}\\
&= \frac{\exp\left(b^{\mathrm{T}}h\right)\exp\left(v^{\mathrm{T}}Wh + a^{\mathrm{T}}v\right)}{\exp\left(b^{\mathrm{T}}h\right)\sum_{\hat{v}} \exp\left(\hat{v}^{\mathrm{T}}Wh + a^{\mathrm{T}}\hat{v}\right)} && \text{(C.34)}\\
&= \frac{\exp\left(\sum_{i=1}^{D}\sum_{j=1}^{M} v_i W_{ij} h_j + \sum_{i=1}^{N} a_i v_i\right)}{\sum_{\hat{v}} \exp\left(\sum_{i=1}^{D}\sum_{j=1}^{M} \hat{v}_i W_{ij} h_j + \sum_{i=1}^{N} a_i \hat{v}_i\right)} && \text{(C.35)}\\
&= \frac{\prod_{i=1}^{D} \exp\left(\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) v_i\right)}{\sum_{\hat{v}} \prod_{i=1}^{D} \exp\left(\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) \hat{v}_i\right)} && \text{(C.36)}\\
&= \frac{\prod_{i=1}^{D} \exp\left(\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) v_i\right)}{\left(1 + \sum_{j=1}^{M} W_{ij}h_j + a_i\right) \sum_{\hat{v}_2} \cdots \sum_{\hat{v}_N} \exp\left(\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) \hat{v}_i\right)} && \text{(C.37)}\\
&= \prod_{i=1}^{N} \frac{\exp\left(\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) v_i\right)}{1 + \exp\left(\sum_{j=1}^{M} W_{ij}h_{jn} + a_i\right)} && \text{(C.38)}\\
&= \prod_{i=1}^{N} p(v_i|h) && \text{(C.39)}
\end{aligned}$$

In the step from Eq. C.36 to Eq. C.36 the sum over the configurations $\hat{v}_1$ is written out and configuration values are inserted. Then this step is repeated for all sums over configurations $\hat{v}_i$. Finally, sampling can be carried out by:

$$p(v_i = 1|h) = \sigma\left(\sum_{j=1}^{M} W_{ij}h_j + a_i\right) \tag{C.40}$$

The model is symmetric, therefore:

$$p(h|v) = \prod_{j=1}^{F} p(h_j|v); \quad p(h_j = 1|v) = \sigma\left(\sum_{i=1}^{N} W_{ij}v_i + b_j\right) \tag{C.41}$$

C.4 Conditional independence and sampling in the EFH model

Given visible units with distributions from a member of the exponential family combined multiplicatively:

$$p(\{v_i\}) = \prod_{i=1}^{D} r_i(v_i) \exp\left(\sum_a \theta_{ia} f_{ia}(v_i) - A_i(\{\theta_{ia}\})\right) \tag{C.42}$$

with A_i being the log-partition function, $f_{ia}(v_i)$ the sufficient statistics and θ_{ia} the natural parameters. Likewise, assume the hidden units stem from a possibly different member of the exponential family and are analogously combined multiplicatively:

$$p(\{h_j\}) = \prod_{j=1}^{F} s_j(h_j) \exp\left(\sum_b \lambda_{jb} g_{jb}(h_j) - B_j(\{\lambda_{jb}\})\right) \tag{C.43}$$

with B_j being the log-partition function, $g_{jb}(h_j)$ the sufficient statistics and λ_{jb} the natural parameters. According to Welling et al. [58], the hidden and visible units distributions can be combined by an energy function as follows:

$$E(v,h) = -\sum_{ia} \theta_{ia} f_{ia}(v_i) - \sum_{jb} \lambda_{jb} g_{jb}(h_j) - \sum_{ijab} W_{ia}^{jb} f_{ia}(v_i) g_{jb}(h_j) \tag{C.44}$$

with W_{ia}^{jb} being the coupling tensor (weight tensor).

Now we will show in detail that the conditional distribution $p(v|h)$ indeed factorizes. We start with the application of basic probability theory laws:[1]

$$p(v|h) = \frac{\exp(-E(v,h))}{\int \exp(-E(v',h))dv'} = \tag{C.45}$$

$$= \frac{\exp\left(\sum_{ia} \theta_{ia} f_{ia}(v_i) + \sum_{jb} \lambda_{jb} g_{jb}(h_j) + \sum_{ijab} W_{ia}^{jb} f_{ia}(v_i) g_{jb}(h_j)\right)}{\int \exp\left(\sum_{ia} \theta_{ia} f_{ia}(v'_i) + \sum_{jb} \lambda_{jb} g_{jb}(h_j) + \sum_{ijab} W_{ia}^{jb} f_{ia}(v'_i) g_{jb}(h_j)\right) dv'} \tag{C.46}$$

$$= \frac{\prod_i \exp\left(\sum_a \theta_{ia} f_{ia}(v_i) + \sum_{jab} W_{ia}^{jb} f_{ia}(v_i) g_{jb}(h_j)\right)}{\int \prod_i \exp\left(\sum_a \theta_{ia} f_{ia}(v'_i) + \sum_{jab} W_{ia}^{jb} f_{ia}(v'_i) g_{jb}(h_j)\right) dv'} \tag{C.47}$$

The $\sum_{jb} \lambda_{jb} g_{jb}(h_j)$ terms cancel out and the sums are transformed into products outside of the exponential function. The following step assumes independence, but since we want to show conditional independence it is ok. [2]

[1] for discrete probability distributions replace the integral with a sum

[2] see e.g. `http://planetmath.org/encyclopedia/MultidimensionalGaussianIntegral.html`

$$= \prod_i^D \frac{\exp\left(\sum_a \theta_{ia} f_{ia}(v_i) + \sum_{jab} W_{ia}^{jb} f_{ia}(v_i) g_{jb}(h_j)\right)}{\int \exp\left(\sum_a \theta_{ia} f_{ia}(v'_i) + \sum_{jab} W_{ia}^{jb} f_{ia}(v'_i) g_{jb}(h_j)\right) dv'_i} \tag{C.48}$$

$$= \prod_i^D \frac{\exp\left(\sum_a \left(\theta_{ia} + \sum_{jb} W_{ia}^{jb} g_{jb}(h_j)\right) f_{ia}(v_i)\right)}{\int \exp\left(\sum_a \left(\theta_{ia} + \sum_{jb} W_{ia}^{jb} g_{jb}(h_j)\right) f_{ia}(v'_i)\right) dv'_i} \tag{C.49}$$

With $\hat{\theta}_{ia} := \theta_{ia} + \sum_{jb} W_{ia}^{jb} g_{jb}(h_j)$:

$$= \prod_i^D \exp\left(\sum_a \hat{\theta}_{ia} f_{ia}(v_i) - \int \sum_a \hat{\theta}_{ia} f_{ia}(v'_i) dv'_i\right) \tag{C.50}$$

$$= \prod_i^D \exp\left(\sum_a \hat{\theta}_{ia} f_{ia}(v_i) - A(\{\hat{\theta}_{ia}\})\right) \tag{C.51}$$

For the last step the identity $\int_y \exp(\sum_a \theta_a f_a(y))dy = \exp(A(\{\theta_a\}))$ was used. It holds, due to the following argumentation:[3]

$$\int_y p(y)dy = \int_y r(y) \exp\left(\sum_a \theta_a f_a(y) - A(\{\theta_a\})\right) dy \tag{C.52}$$

$$= \exp\left(-A(\{\theta_a\})\right) \int_y r(y) \exp\left(\sum_a \theta_a f_a(y)\right) dy = 1 \tag{C.53}$$

and therefore:

$$\int_y r(y) \exp\left(\sum_a \theta_a f_a(y)\right) dy = \exp\left(A(\{\theta_a\})\right) \tag{C.54}$$

The model is symmetric, consequently the derivation for $p(h|v)$ is analogue.

C.5 Conditional independence and sampling in the RSM model

Assume the following energy function given by Salakhutdinov and Hinton [47]:

$$E(V,h) = -\sum_{k=1}^K h^{\mathrm{T}} W^k V^k - \left(a^k\right)^{\mathrm{T}} V^k - h^{\mathrm{T}} b \tag{C.55}$$

or equivalently:

$$E(V,h) = -\sum_{i=1}^D \sum_{j=1}^F \sum_{k=1}^K h_j W_{ij}^k V_i^k - \sum_{i=1}^D \sum_{k=1}^K V_i^k a_i^k - \sum_{j=1}^F h_j b_j \tag{C.56}$$

[3]see e.g. `http://www.cs.columbia.edu/~jebara/4771/tutorials/lecture12.pdf`, page 2

with binary hidden units $h_j \in \{0,1\}$, $j = 1\dots,F$, a binary dictionary matrix $V_i^k \in \{0,1\}$, $i = 1,\dots,D$, $k = 1,\dots,K$ and a coupling tensor W_{ij}^k (or using weight sharing a coupling matrix W_j^k). The conditional distribution $p(V|h)$ factorizes into a product of softmax functions as follows:

$$p(V|h) = \frac{\exp -E(V,h)}{\sum_{\tilde{V}} \exp(-E(\tilde{V},h))} \tag{C.57}$$

$$= \frac{\exp(\sum_{k=1}^K h^{\mathrm{T}} W^k V^k + \left(a^k\right)^{\mathrm{T}} V^k + h^{\mathrm{T}} b)}{\sum_{\tilde{V}} \exp(\sum_{k=1}^K h^{\mathrm{T}} W^k \tilde{V}^k + \left(a^k\right)^{\mathrm{T}} \tilde{V}^k + h^{\mathrm{T}} b)} \tag{C.58}$$

$$= \frac{\exp(\sum_{k=1}^K h^{\mathrm{T}} W^k V^k + \left(a^k\right)^{\mathrm{T}} V^k)}{\sum_{\tilde{V}} \exp(\sum_{k=1}^K h^{\mathrm{T}} W^k \tilde{V}^k + \left(a^k\right)^{\mathrm{T}} \tilde{V}^k)} \tag{C.59}$$

$$= \frac{\exp(\sum_{i=1}^D \sum_{k=1}^K h^{\mathrm{T}} W_i^k V_i^k + a_i^k V_i^k)}{\sum_{\tilde{V}} \exp(\sum_{i=1}^D \sum_{k=1}^K h^{\mathrm{T}} W_i^k \tilde{V}_i^k + a_i^k \tilde{V}_i^k)} \tag{C.60}$$

$$= \frac{\prod_{i=1}^D \prod_{k=1}^K \exp((h^{\mathrm{T}} W_i^k + a_i^k) V_i^k)}{\sum_{\tilde{V}} \exp(\sum_{i=1}^D \sum_{k=1}^K (h^{\mathrm{T}} W_i^k + a_i^k) \tilde{V}_i^k)} \tag{C.61}$$

The denominator of Eq. C.61 can be reformulated as follows:

$$\sum_{\tilde{V}} \exp\left(\sum_{i=1}^D \sum_{k=1}^K (h^{\mathrm{T}} W_i^k + a_i^k) \tilde{V}_i^k\right) \tag{C.62}$$

$$= \sum_{\tilde{V}} \prod_{i=1}^D \prod_{k=1}^K \exp\left(\left(h^{\mathrm{T}} W_i^k + a_i^k\right) \tilde{V}_i^k\right) \tag{C.63}$$

$$= \sum_{\tilde{V}} \prod_{i=1}^D \prod_{k=1}^K \exp\left(\left(\sum_{j=1}^F h_j W_{ij}^k + a_i^k\right) \tilde{V}_i^k\right) \tag{C.64}$$

$$= \sum_{\tilde{V}_1} \sum_{\tilde{V}_2} \cdots \sum_{\tilde{V}_D} \prod_{i=1}^D \prod_{k=1}^K \exp\left(\left(\sum_{j=1}^F h_j W_{ij}^k + a_i^k\right) \tilde{V}_i^k\right) \tag{C.65}$$

$$= \left(\sum_{\tilde{V}_1} \prod_{k=1}^K \exp\left(\left(\sum_{j=1}^F h_j W_{1j}^k + a_1^k\right) \tilde{V}_1^k\right)\right) \sum_{\tilde{V}_2} \cdots \sum_{\tilde{V}_D} \prod_{i=2}^D \prod_{k=1}^K \exp\left(\left(\sum_{j=1}^F h_j W_{ij}^k + a_i^k\right) \tilde{V}_i^k\right) \tag{C.66}$$

$$\overset{*}{=} \left(\sum_{q=1}^K \exp\left(\sum_{j=1}^F h_j W_{1j}^q + a_1^q\right)\right) \sum_{\tilde{V}_2} \cdots \sum_{\tilde{V}_D} \prod_{i=2}^D \prod_{k=1}^K \exp\left(\left(\sum_{j=1}^F h_j W_{ij}^k + a_i^k\right) \tilde{V}_i^k\right) \tag{C.67}$$

$$= \prod_{i=1}^D \sum_{q=1}^K \exp\left(\sum_{j=1}^F h_j W_{ij}^q + a_i^q\right) \tag{C.68}$$

The step from Eq. C.66 to C.67 ($\overset{*}{=}$) is valid, since with $r_i^k := \sum_{j=1}^F h_j W_{ij}^k + a_i^k$ one can see that the following holds:

$$\sum_{\tilde{V}_i} \prod_{k=1}^K \exp\left(r_i^k \tilde{V}_i^k\right) = \tag{C.69}$$

$$\Big[\underbrace{\exp\left(r_i^1 1\right)}_{=\exp(r_i^1)} \underbrace{\exp\left(r_i^2 0\right)}_{=1} \dots \underbrace{\exp\left(r_i^K 0\right)}_{=1}\Big] + \dots + \tag{C.70}$$

$$+\Big[\underbrace{\exp\left(r_i^1 0\right)}_{=1} \underbrace{\exp\left(r_i^2 0\right)}_{=1} \dots \underbrace{\exp\left(r_i^K 1\right)}_{=\exp(r_i^K)}\Big] =$$

$$\sum_{q=1}^K \exp\left(r_i^q\right) \tag{C.71}$$

The $\tilde{V}_i$ define a 1-of-K coding scheme ($\tilde{V}_i^k \in \{0, 1\}$ and $\sum_{k=1}^K \tilde{V}_i^k = 1$). However, all K possible configurations must be taken into account, which is reflected by the term $\sum_{q=1}^K \exp(r_i^q)$. Inserted into Eq. C.61:

$$\frac{\prod_{i=1}^D \prod_{k=1}^K \exp\left(\left(h^\mathrm{T} W_i^k + a_i^k\right) V_i^k\right)}{\prod_{i=1}^D \sum_{q=1}^K \exp\left(\sum_{j=1}^F h_j W_{ij}^q + a_i^q\right)} = p(V_i|h) \tag{C.72}$$

Thus:

$$p(V_i^k = 1|h) = \frac{\prod_{k=1}^K \exp\left(\left(h^\mathrm{T} W_i^k + a_i^k\right) V_i^k\right)}{\sum_{q=1}^K \exp\left(\sum_{j=1}^F h_j W_{ij}^q + a_i^q\right)} \tag{C.73}$$

$$= \frac{\exp\left(h^\mathrm{T} W_i^k + a_i^k\right)}{\sum_{q=1}^K \exp\left(\sum_{j=1}^F h_j W_{ij}^q + a_i^q\right)} \tag{C.74}$$

holds proposed. And for $p(h|V)$ we get:

$$p(h|V) = \frac{\exp -E(V, h)}{\sum_{\tilde{h}} \exp(-E(V, \tilde{h}))} \tag{C.75}$$

$$= \frac{\exp\left(\sum_{k=1}^K h^\mathrm{T} W^k V^k + \left(a^k\right)^\mathrm{T} V^k + h^\mathrm{T} b\right)}{\sum_{\tilde{h}} \exp\left(\sum_{k=1}^K \tilde{h}^\mathrm{T} W^k V^k + \left(a^k\right)^\mathrm{T} V^k + \tilde{h}^\mathrm{T} b\right)} \tag{C.76}$$

$$= \frac{\prod_{j=1}^F \exp\left(\sum_{i=1}^D \sum_{k=1}^K \left(b_j + W_{ij}^k v_i^k\right) h_j\right)}{\sum_{\tilde{h}} \prod_{j=1}^F \exp\left(\sum_{i=1}^D \sum_{k=1}^K \left(b_j + W_{ij}^k v_i^k\right) \tilde{h}_j\right)} \tag{C.77}$$

$$= \prod_{j=1}^F \frac{\exp\left(\sum_{i=1}^D \sum_{k=1}^K \left(b_j + W_{ij}^k v_i^k\right) h_j\right)}{\sum_{\tilde{h}_j} \exp\left(\sum_{i=1}^D \sum_{k=1}^K \left(b_j + W_{ij}^k v_i^k\right) \tilde{h}_j\right)} \tag{C.78}$$

$$\tag{C.79}$$

Consequently, sampling is easy, too:

$$p(h_j = 1|V) = \frac{\exp\left(\sum_{i=1}^{D}\sum_{k=1}^{K}\left(b_j + W_{ij}^k v_i^k\right)\right)}{1 + \exp\left(\sum_{i=1}^{D}\sum_{k=1}^{K}\left(b_j + W_{ij}^k v_i^k\right)\right)} \tag{C.80}$$

$$= \sigma\left(b_j + \sum_{i=1}^{D}\sum_{k=1}^{K} W_{ij}^k V_i^k\right) \tag{C.81}$$

with $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$ being the logistic sigmoid function.

Using the following energy function with $\hat{V}^k := \sum_{i=1}^{D} V_i^k$, i.e. with weight sharing in action which leads to the RSM model according Salakhutdinov and Hinton [47] (with omitted index i at W compared to Eq. C.56):

$$E(\hat{V}, h) = -\sum_{j=1}^{F}\sum_{k=1}^{K} h_j W_j^k \hat{V}^k - \sum_{k=1}^{K} \hat{V}^k a^k - D\sum_{j=1}^{F} h_j b_j \tag{C.82}$$

one obtains with a derivation similar to the above:

$$p(h_j = 1|\hat{V}) = \sigma\left(D b_j + \sum_{k=1}^{K} W_j^k \hat{V}^k\right) \tag{C.83}$$

and

$$p(\hat{V}^k = 1|h) = \frac{\exp\left(a^k + \sum_{j=1}^{F} h_j W_j^k\right)}{\sum_{q=1}^{K}\exp\left(a^q + \sum_{j=1}^{F} h_j W_j^q\right)} \tag{C.84}$$

Sample D times using Eq. C.84.

Bibliography

[1] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, *NP-hardness of Euclidean sum-of-squares clustering*, Mach. Learn., 75 (2009), pp. 245–248.

[2] D. Arthur and S. Vassilvitskii, *k-means++: The Advantages of Careful Seeding*, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '07, Philadelphia, PA, USA, 2007, Society for Industrial and Applied Mathematics, pp. 1027–1035.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, *SURF: Speeded-Up Robust Features*, Computer Vision and Image Understanding, 110 (2008), pp. 346–359.

[4] S. Bengio, F. Pereira, Y. Singer, and D. Strelow, *Group Sparse Coding*, in Advances in Neural Information Processing Systems 22, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, eds., 2009, pp. 82–89.

[5] Y. Bengio, *Learning deep architectures for AI*, Foundations and Trends in Machine Learning, 2 (2009), pp. 1–127. Also published as a book. Now Publishers, 2009.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, *Greedy Layer-Wise Training of Deep Networks*, in Advances in Neural Information Processing Systems 19 (NIPS'06), 2006, pp. 153–160.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer, 1 ed., 2007.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, *Latent Dirichlet Allocation*, J. Mach. Learn. Res., 3 (2003), pp. 993–1022.

[9] L. Bottou and Y. Bengio, *Convergence Properties of the K-Means Algorithms*, in Advances in Neural Information Processing Systems 7, MIT Press, 1995, pp. 585–592.

[10] G. J. Burghouts, A. W. M. Smeulders, and J.-M. Geusebroek, *The Distribution Family of Similarity Distances*, in NIPS, 2007.

[11] N. Dalal and B. Triggs, *Histograms of Oriented Gradients for Human Detection*, in CVPR (1), 2005, pp. 886–893.

[12] C. Ding and X. He, *K-means Clustering via Principal Component Analysis*, in Proceedings of the 21st International Conference on Machine Learning, ICML '04, New York, NY, USA, 2004, ACM, pp. 29–.

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (voc2007) results*. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[14] Y. FREUND AND D. HAUSSLER, *Unsupervised learning of distributions on binary vectors using two layer networks*, tech. report, University of California at Santa Cruz, 1994.

[15] P. V. GEHLER, A. D. HOLUB, AND M. WELLING, *The Rate Adapting Poisson Model for Information Retrieval and Object Recognition*, in Proceedings of 23rd International Conference on Machine Learning (ICML 2006, ACM Press, 2006, p. 2006.

[16] G. HINTON, *A Practical Guide to Training Restricted Boltzmann Machines (Version 1).* online, August 2010.

[17] G. HINTON AND R. SALAKHUTDINOV, *Reducing the Dimensionality of Data with Neural Networks*, Science, 313 (2006), pp. 504 – 507.

[18] G. E. HINTON, *Training Products of Experts by Minimizing Contrastive Divergence*, Neural Comput., 14 (2002), pp. 1771–1800.

[19] G. E. HINTON, S. OSINDERO, AND Y.-W. TEH, *A fast learning algorithm for deep belief nets*, Neural Comput., 18 (2006), pp. 1527–1554.

[20] T. HOFMANN, *Probabilistic Latent Semantic Indexing*, in Proceedings of the 22nd annual international ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '99, New York, NY, USA, 1999, ACM, pp. 50–57.

[21] Y. G. JIANG, J. YANG, C. W. NGO, AND A. G. HAUPTMANN, *Representations of Keypoint-Based Semantic Concept Detection: A Comprehensive Study*, Multimedia, IEEE Transactions on, 12 (2009), pp. 42–53.

[22] T. JOACHIMS, *Making Large-Scale SVM Learning Practical*, in Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. J. Burges, and A. Smola, eds., Cambridge, MA, USA, 1999, MIT Press.

[23] H. KANG, M. HEBERT, AND T. KANADE, *Image matching with distinctive visual vocabulary*, in Proceedings of the 2011 IEEE Workshop on Applications of Computer Vision (WACV), WACV '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 402–409.

[24] Y. KE AND R. SUKTHANKAR, *PCA-SIFT: A More Distinctive Representation for Local Image Descriptors*, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, 2 (2004), pp. 506–513.

[25] A. KRIZHEVSKY, *Learning Multiple Layers of Features from Tiny Images*, master thesis, University of Toronto, 2009.

[26] S. LACOSTE-JULIEN, F. SHA, AND M. I. JORDAN, *DiscLDA: Discriminative Learning for Dimensionality Reduction and Classification.*

[27] S. P. LLOYD, *Least Squares Quantization in PCM*, IEEE Transactions on Information Theory, 28 (1982), pp. 129–137.

[28] D. G. LOWE, *Object Recognition from Local Scale-Invariant Features*, in Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2, ICCV '99, Washington, DC, USA, 1999, IEEE Computer Society, pp. 1150–.

[29] D. G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints.*, International Journal of Computer Vision, 60 (2004), pp. 91–110.

[30] J. B. MacQueen, *Some Methods for Classification and Analysis of MultiVariate Observations*, in Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, L. M. L. Cam and J. Neyman, eds., vol. 1, University of California Press, 1967, pp. 281–297.

[31] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, *Online dictionary learning for sparse coding*, in Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, New York, NY, USA, 2009, ACM, pp. 689–696.

[32] C. D. Manning, P. Raghavan, and H. Schuetze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[33] T. K. Marks and J. R. Movellan, *Diffusion Networks, Products of Experts, and Factor Analysis*, in in Proc. 3rd Int. Conf. Independent Component Anal. Signal Separation, 2001, pp. 481–485.

[34] W. S. McCulloch and W. Pitts, *A Logical Calculus of Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, 5 (1943), pp. 115–133. Reprinted in *Neurocomputing: Foundations of Research, ed. by J. A. Anderson and E Rosenfeld. MIT Press 1988.*

[35] K. Mikolajczyk and C. Schmid, *A performance evaluation of local descriptors*, IEEE Transactions on Pattern Analysis & Machine Intelligence, 27 (2005), pp. 1615–1630.

[36] V. Mnih, *CUDAMat: A CUDA-based matrix class for Python*, Tech. Report UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

[37] J. A. Nelder and R. W. M. Wedderburn, *Generalized linear models*, Journal of the Royal Statistical Society, Series A, General, 135 (1972), pp. 370–384.

[38] D. Nistér and H. Stewénius, *Scalable Recognition with a Vocabulary Tree*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2, June 2006, pp. 2161–2168.

[39] A. D. Peterson, A. P. Ghosh, and R. Maitra, *A systematic evaluation of different methods for initializing the K-means clustering algorithm*, Knowledge Creation Diffusion Utilization, (2010), pp. 1–11.

[40] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, *Object Retrieval with Large Vocabularies and Fast Spatial Matching*, in IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[41] J. Philbin, M. Isard, J. Sivic, and A. Zisserman, *Descriptor Learning for Efficient Retrieval*, in European Conference on Computer Vision, 2010.

[42] P. Resnik and E. Hardisty, *Gibbs Sampling for the Uninitiated*, Tech. Report LAMP-TR-153, University of Maryland, College Park, 2010.

[43] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Book, 1962.

[44] N. L. Roux, N. Heess, J. Shotton, and J. M. Winn, *Learning a Generative Model of Images by Factoring Appearance and Shape*, Neural Computation, 23 (2011), pp. 593–650.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, MIT Press, Cambridge, MA, USA, 1986, pp. 318–362.

[46] R. Salakhutdinov and G. Hinton, *Semantic Hashing*, in SIGIR Workshop on Information Retrieval and Applications of Graphical Models, 2007.

[47] R. Salakhutdinov and G. Hinton, *Replicated Softmax: an Undirected Topic Model*, in Neural Information Processing Systems 23, 2010.

[48] R. Salakhutdinov and I. Murray, *On the Quantitative Analysis of Deep Belief Networks*, in Proceedings of the International Conference on Machine Learning, vol. 25, 2008.

[49] M. S. Sarfraz and O. Hellwich, *Head Pose Estimation in Face Recognition Across Pose Scenarios*, in VISAPP (1), 2008, pp. 235–242.

[50] S. Schroeder, *Interaktion gedächtnis- und erklärungs-basierter Verarbeitungsprozesse bei der pronominalen Auflösung*, PhD thesis, Universität Köln, 2007.

[51] D. Sculley, *Web-scale k-means clustering*, in Proceedings of the 19th international conference on World wide web, WWW '10, New York, NY, USA, 2010, ACM, pp. 1177–1178.

[52] J. Sivic and A. Zisserman, *Video Google: A text retrieval approach to object matching in videos*, in Proceedings of the International Conference on Computer Vision, vol. 2, Oct. 2003, pp. 1470–1477.

[53] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, MIT Press, Cambridge, MA, USA, 1986, pp. 194–281.

[54] T. Tieleman, *Gnumpy: an easy way to use GPU boards in Python*, Tech. Report UTML TR 2010-002, University of Toronto, Department of Computer Science, 2010.

[55] L. Van Der Maaten and G. Hinton, *Visualizing Data using t-SNE*, Journal of Machine Learning Research, 9 (2008), pp. 2579–2605.

[56] A. Vedaldi., *An open implementation of the SIFT detector and descriptor*, 2007.

[57] A. Vedaldi, *VLfeat SIFT Tutorial.* Online, June 2011. `http://www.vlfeat.org/overview/sift.html`.

[58] M. Welling, M. Rosen-Zvi, and G. Hinton, *Exponential Family Harmoniums with an Application to Information Retrieval*, Training, 17 (2005), pp. 1481–1488.

[59] B. Widrow and M. E. Hoff, *Adaptive Switching Circuits*, in Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, 1960, pp. 96–104.

[60] S. A. J. Winder and M. Brown, *Learning Local Image Descriptors*, in CVPR, 2007, pp. 1–8.

[61] F. Wood, *Training Products of Experts by Minimizing Contrastive Divergence from Geoffrey E. Hinton presented by Frank Wood.* http://www.cs.brown.edu/ fwood/tutorials/ContrastiveDivergenceSlides.ppt.

[62] E. P. Xing, R. Yan, and A. G. Hauptmann, *Mining Associated Text and Images with Dual-Wing Harmoniums*, in In Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-2005, AUAI press, 2005.

[63] J. Yang, R. Yan, Y. Liu, and E. P. Xing, *Harmonium Models for Video Classification*, Statistical Analysis and Data Mining, 1 (2008), pp. 23–37.

[64] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon, *Spectral Relaxation for K-means Clustering.*, in NIPS'01, 2001, pp. 1057–1064.

[65] H. Zhang, T. W. S. Chow, and M. K. M. Rahman, *A new dual wing harmonium model for document retrieval*, Pattern Recogn., 42 (2009), pp. 2950–2960.

[66] X. Zhou, K. Yu, T. Zhang, and T. S. Huang, *Image Classifcation using Super-Vector Coding of Local Image Descriptors*, in Proceedings of the 11th European Conference on Computer Vision: Part V, ECCV'10, Berlin, Heidelberg, 2010, Springer-Verlag, pp. 141–154.