Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Jan Hajič

Matching Images to Texts

Institute of Formal and Applied Linguistics (ÚFAL)

Supervisor of the master thesis:	RNDr. Pavel Pecina, Ph.D.
Study programme:	Computer Science
Specialization:	Mathematical Linguistics

Prague 2014

I would like to thank all those that helped deliver this thesis into the world. At the forefront:

- My advisor, RNDr. Pavel Pecina, for his leadership, support and faith,
- My family, who have been incredibly supportive,
- Adélka Venclová, for her undying patience.

Even the mere fact that this thesis has been written is as much their achievement as mine. Perhaps more.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In .... date .... signature of the author

Název práce: Hledání obrázků k textům

Autor: Bc. Jan Hajič

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: RNDr. Pavel Pecina, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Vytváříme společný pravděpodobnostní model textu a obrázků pro úlohu automatického přiřazování ilustračních fotografií k novinovým článkům. Přistupujeme k úloze z hlediska *učení reprezentací*: chceme nalézt společnou reprezentaci textu i obrázků nezávislou na vlastnostech jednotlivých modalit, podobně jako multimodální hluboký Boltzmannův stroj Srivastavy a Salakhutdinova. Vstupní obrázky reprezentujeme pomocí předposlední vrstvy konvoluční neuronové sítě Krizhevského a kol., state-of-the-art reprezentace obrázků na základě jejich obsahu. Vytvořili jsme knihovnu *Safire* pro hluboké učení a správu multimodálních experimentů. Úspěšný vyhledávací systém se nám vyvinout nepodařilo, kvůli obtížnému trénování neuronových sítí na velmi řídkých textových datech. Porozuměli jsme však povaze těchto potíží tak, že věříme, že v navazující práci můžeme lepších výsledků dosáhnout.

Klíčová slova: hluboké učení, neuronové sítě, multimodální model, učení reprezentací

Title: Matching Images to Texts

Author: Bc. Jan Hajič

Department: Institute of Formal and Applied Linguistics (ÚFAL)

Supervisor: RNDr. Pavel Pecina, Ph.D., Institute of Formal and Applied Linguistics (ÚFAL)

Abstract: We build a joint multimodal model of text and images for automatically assigning illustrative images to journalistic articles. We approach the task as an unsupervised *representation learning* problem of finding a common representation that abstracts from individual modalities, inspired by multimodal Deep Boltzmann Machine of Srivastava and Salakhutdinov. We use state-of-the-art image content classification features obtained from the Convolutional Neural Network of Krizhevsky et al. as input “images” and entire documents instead of keywords as input texts. A deep learning and experiment management library *Safire* has been developed. We have not been able to create a successful retrieval system because of difficulties with training neural networks on the very sparse word observation. However, we have gained substantial understanding of the nature of these difficulties and thus are confident that we will be able to improve in future work.

Keywords: deep learning, neural networks, text-image models, representation learning

Contents

1. Introduction

An image is allegedly worth a thousand words. In this thesis, we attempt to find images that are worth the *given* words, a news article: we attempt to automatically retrieve appropriate illustrative images for certain types of journalistic texts.

The instant expressive power of images is for journalism – with its focus on impact, on getting a point across in shorthand – a perfect match. Examples where images manipulate the impact of texts and vice versa are aplenty: both high-profile iconic shots and even more so cases of sustained coupling of certain topics with specific imagery, where various propaganda and marketing campaigns are prime examples, with profound effect on the recipients.

FACE THE FACTS!
"Reach for a Lucky instead"
"It's toasted"
20,679 Physicians say "LUCKIES are less irritating"
"It's toasted"
To keep a slender figure No one can deny...
Reach for a LUCKY instead of a sweet
"It's toasted"
Marlboro

Figure 1.1: Tobacco advertising: associating positive imagery with cigarettes

Figure 1.2: Anti-smoking advertisments

While personal creativity of the journalist is certainly involved in choosing an appropriate image to accompany his or her text, we hypothesize that the relationship of text and illustrative photos is predictable enough to allow us to *automatically assign appropriate images to journalistic texts.*

That is our primary goal: **to build a system that will automatically assign illustrative images to texts**. It should be made clear right away that in this respect, **we did not succeed.** This work has negative results.

To gain better insight into *why* illustrative images are matched to the texts they accompany, we will also create a dataset with manual annotations of appropriateness that should help towards future understanding of what the nature of the text-illustrative image relationship is.

We approach the problem as a *representation learning* task, where the goal is to learn a representation that abstracts from the individual modalities to a level where the representation can be shared across the texts and images, therefore obtaining a joint distribution that can be conditioned on either modality and used to generate an image given a text. State-of-the-art results have been obtained on multimodal tasks by using a topic model [26] or more recently using a *distributed representation* obtained using *deep learning* by Srivasta and Salakhutdinov [73].

While very similar in approach to Srivastava and Salakhutdinov's deep network, our work is different in two important ways. First, while their work uses image tags as the text modality, we use entire documents (similar to Feng and Lapata). Second, we do not learn image features from scratch, but use a state-of-the-art model used for the Imagenet-1000 image content classification challenge by Krizhevsky et al. [45] to transform our raw image data to an advanced representation and re-purpose this layer for retrieval. Also, our data exhibits a different, less straightforward relationship between the text and image modalities than [73]. Srivastava and Salakhutdinov used the Flickr dataset where the text modality for a given image is added by the uploader of the photo for that specific picture to describe its content. On the other hand, in our data, images were manually assigned[1] to accompany the already finished documents from a pool of candidate images. (There are other differences as well; refer to the full experimental setup described in chapter 6.)

We chose to pursue the distributed representation path, also in order to gain experience with the (still relatively) novel field of *deep learning*, the building of multi-layer *neural network* models. The bulk of the Related Work chapter (3) will be dedicated to a review of applicable methods and results in deep learning.

As opposed to models like the popular Latent Dirichlet Allocation (LDA) [9], distributed representations of text allow the learned "topics" to *interact*. In a distributed topic model, the presence of topics "media", "agriculture" and "StB" in the same news article may dramatically increase the chances of generating also the word "Babiš", much more so than a marginalization over the topic proportions. When layered, distributed topics are also a *compact* way of expressing such interactions: an LDA topic model would need a topic for each combination of media, business and StB to express the large probability of Babiš when all three topics are present but not when only one or two are. Neural network distributed representation are also able to model *inhibitory* effects directly.

Representation learning is one of the leading perspectives on neural network models today; for an extensive review, refer to the work of Bengio, Courville and Vincent from 2012 [4]. It has been shown that after learning a good representation for the input data, one network can be re-purposed for various other supervised tasks. For Natural Language Processing, the seminal work in this respect is Collobert and Weston's SENNA framework [18] which can do morphological tagging, syntactic parsing, named entity recognition and other tasks above representations shared across tasks.

We can also call representation learning *feature discovery*: the network is able to learn what constitutes good features for the given data. In this respect, neural networks, especially their deep variants, may become an alternative to hand-

[1]News portals employ a specialist or two specifically to assign pictures to articles; the journalists that write the articles do not do this.

crafting features. (This is something any representation learning model can do, and practically any model can be interpreted as learning a representation, but well-designed deep neural networks seem to be particularly good at it.)

The notion of what a good representation is is formalized by the appropriate optimization objective. Often, objective functions measure both the network's ability to generalize and some interesting property of the representation itself, like sparsity or other regularization; in models where the input and the learned features have a probabilistic interpretation, various priors can be utilized.

The distributed representations learned by neural networks have also been successfully applied to many classification tasks, where a linear classifier such as a Support Vector Machine is trained on top of the final layer's representation of the data [73], or even used directly for discrimination [46].

While notable advances have been made in learning distributed representations for *documents* [70] [75] [35], it is still much of an open problem [21].

In the rest of the introduction, we will shortly introduce neural networks and deep learning. The rest of the work is organized as follows:

- Chapter 2 introduces the common families of deep learning models,
- Chapter 3 reviews related work in deep learning and multimodal models,[2]
- Chapter 4 describes the dataset and manual annotations,
- Chapter 5 describes preprocessing of the image and text data,
- Chapter 6 describes the models and learning setups we used,
- Chapter 7 evaluates our model's results,
- Chapter 8 briefly describes the SAFIRE library we implemented for this work,
- Chapter 9 discusses the experiment results and future work.

1.1 Neural networks

Neural networks (NNs) are machine learning models inspired by the functioning of the brain and central nervous system in animals. The networks consist of *neurons*, which are organized into a progression of *layers*. The layers are interconnected: every neuron of the i-th layer sends signal to every neuron of the next layer, inspired by the action of axons[3] and dendrites[4] in the biological nervous system. Some connections are stronger than others – each connection is associated with its *weight*. Neurons are *activated*: the activation of a neuron represents the strength of the signal passing through it, again mimicking the activation of biological neurons. When the signal travels only in the direction from the input to the output layer, we call the model a *feedforward* neural network.

The main appeal of neural networks is the possibility of building a *deep* network, with several hidden layers stacked on top of each other. Each layer should learn a new representation based on the previous layer. The human brain is known to function this way, at least to a significant extent; the V1 visual cortex

[2]This is a somewhat unorthodox position of the Related work chapter; however, we feel that without first describing at least the usual models and introducing the terminology, the value of the related work will not be sufficiently clear.

[3]`https://en.wikipedia.org/wiki/Axon`

[4]`https://en.wikipedia.org/wiki/Dendrite`

is a prime example of layer-wise organization of neurons in the animal brain [68] where the layers progress from neurons that work like edge detectors (active on seeing an edge at a particular angle in the field of vision) all the way to layers that detect high-level features such as faces. With a deep architecture, it is possible to represent such complex features more compactly – with fewer parameters – than in a shallow model with equal expressive power.

A simple neural network with one hidden layer can be visualized like this:

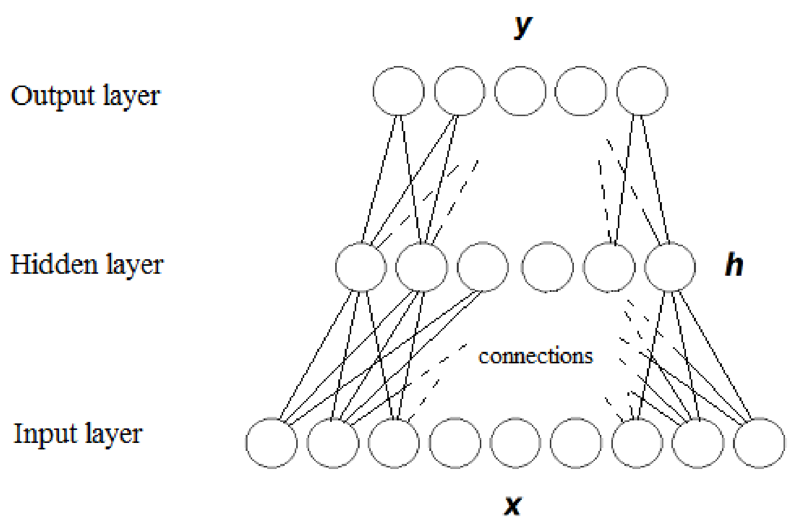


Figure 1.3: Simple feedforward network

Each neuron in a layer represents one dimension of the vector space the layer occupies. One layer is the *input* layer, where neurons correspond to features of the input data; another is the *output* layer, which represents the solutions to the task to which the neural network model has been applied. (For instance, for a 10-class classification task with 1000 features, the network will have 1000 neurons in the input layer and 10 in the output layer.) The other layers are called *hidden*.

Signal travels from one layer to the next in two stages. First, each neuron on the receiving end computes the sum of signals from incoming connections. This signal can also be modified by a *bias* term: an additive constant for each neuron. Second, it applies a nonlinearity, usually sigmoid, to compute its activation. This nonlinearity is called the *activation function*. Typically, activation of neurons is constrained by the activation function to lie between 0 and 1, or between -1 and 1. The activation is then propagated to the next layer's neurons and, together with the weights of the outgoing connections, contributes to the activations of the next layer's neurons.

Formally, this process can be described as

$$y_j = \sigma\left(\sum_{i \in L} x_i * W_{ij} + b_j\right) \tag{1.1}$$

where:

- y_j is the activation of the j-th neuron in layer $L+1$,

- σ is the L-th layer activation function,
- x_i is the activation of the i-th neuron in layer L,
- W_{ij} is the strength of the connection from neuron i of L to neuron j of $L+1$,
- b_j is the bias of neuron j of $L+1$.

The equation can be rewritten for the activation of the entire layer:

$$\mathbf{y} = \sigma\left(\mathbf{W}^T\mathbf{x} + \mathbf{b}\right) \tag{1.2}$$

where $\mathbf{y}$ is the vector $y_1 \dots y_{\|L+1\|}$ of output activations, $\mathbf{x}$ is analogously the vector of input activations, W_{ij} is the j-th cell of the i-th row of the weight matrix $\mathbf{W}$ and $\mathbf{b}$ is the vector of $L+1$-th layer biases. The activation function σ is applied element-wise to the linear activation $\mathbf{W}^T\mathbf{x} + \mathbf{b}$.

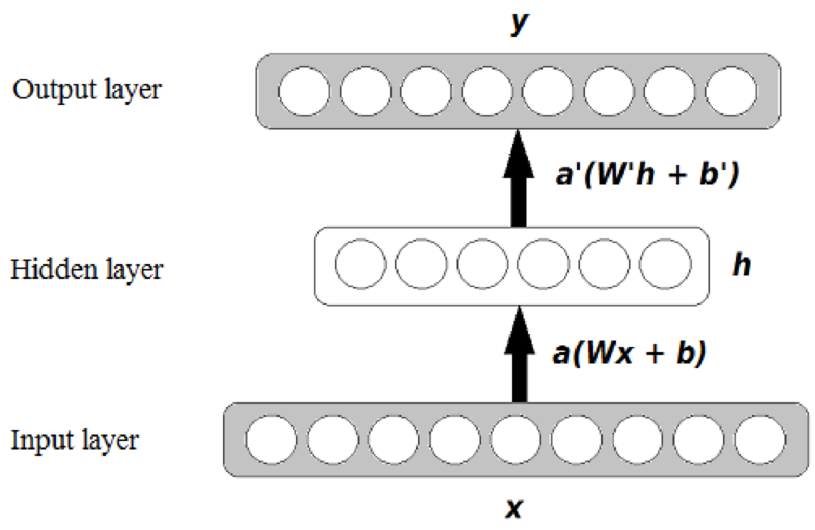


Figure 1.4: Simple feedforward network visualization, "collapsed" connections. The grey outlines represent visible variables, modelled after standard probabilistic graphical model notation.

Typical activation functions are:

- The standard sigmoid: $\sigma(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$, which constrains activations to $(0, 1)$;
- Hyperbolic tangent: $\sigma(\mathbf{x}) = tanh(\mathbf{x})$, which constrains activations to $(-1, 1)$;
- Softmax: $\sigma(x_i) = \frac{e^{W_i x + b_i}}{\sum_{j\in\mathbf{L}} e^{W_j x + b_j}}$, which can be interpreted as the categorical probability of the i-th label being the true label for the given data point,
- ReLU: $\sigma(x) = max(0, x)$. This non-sigmoid activation has recently received much attention after [45] and is currently considered a good choice for supervised learning. Other such as *noisy* ReLUs ($\sigma(x) = max(0, x + \mathcal{N}(0, 1))$)) have been proposed [57].

Neurons can be connected to each other in other connectivity patterns. While most often each neuron of a layer is connected to all neurons in adjacent layer(s), this is, not always the case: Convolutional Neural Networks, which have recently improved state-of-the-art on many image processing tasks, utilize a more complex connectivity pattern with extensive weight sharing.[5]

1.2 Neural network training

We will begin with a canonical example: how to construct a neural network for handwritten digit recognition. Our data will be the famous MNIST dataset of LeCunn [47].

The inputs are 28x28-pixel images, where each pixel is represented by its brightness. We are supposed to classify the images into one of 10 classes (one class per digit). Therefore, the input layer will have 784 neurons and the output layer 10. The true classification will be represented as a vector of 9 zeros and a 1 at position j that corresponds to the true class of the data point. (For images of 0's, the output will be $(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, for 9's, it will be $(0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$, etc.) We are free to choose the number and sizes of hidden layers; for this first example, we will use no hidden layers at all and connect the input layer directly to the output layer.

To obtain a prediction, we will simply take the label corresponding to the most active neuron in the output layer. Training will attempt to minimize a *loss function* between the output layer activations and the vector corresponding to the true class of the data point. We will choose our loss function later.

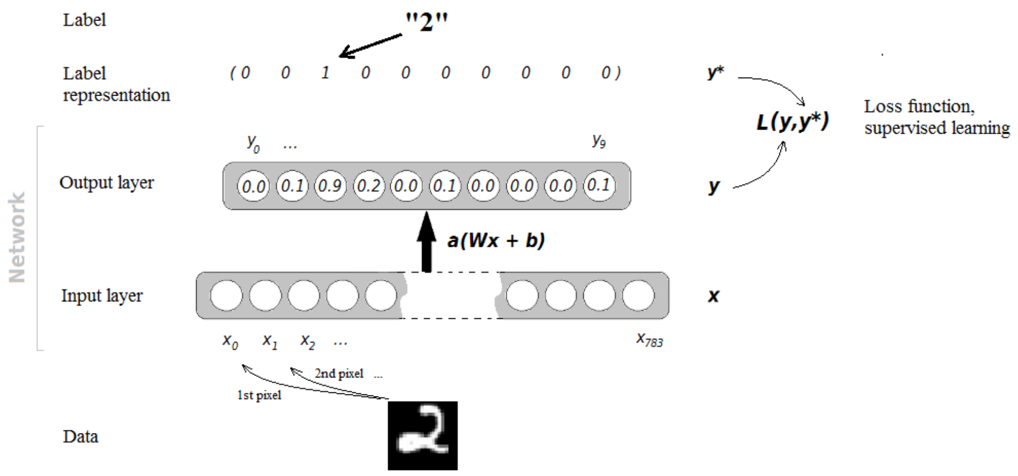


Figure 1.5: A visualization of the classification network action. The loss function acts as the training criterion.

One activation function we can use is the *softmax* function, which can be straightforwardly interpreted to compute class membership probability. When Y is a label that takes on a value $1 \ldots k$, for an output layer with k neurons (k-class classification), we get:

[5] Unrestricted Boltzmann Machines or Hopfield nets would be examples of other connectivity patterns, although they are currently only rarely studied, as their expressive power is more limited.

$$softmax_i(\mathbf{Wx} + \mathbf{b}) = \frac{e^{W_i\mathbf{x}+b_i}}{\sum_{j\in 1\ldots k} e^{W_j\mathbf{x}+b_j}} \tag{1.3}$$
$$= P(Y = i|\mathbf{x}, \mathbf{W}, \mathbf{b}) \tag{1.4}$$

We thus obtain the Multinomial Logistic Regression model. (We do not *have* to use the softmax function and the network would still work, but the interpretation it has is natural for classification tasks.)

Because of our choice of activation function, we can straightforwardly compute *negative log likelihood* of the training data labels. It is a good choice for a loss function, because it measures how our model distributes probability mass for output classes to the correct areas of the space in which the data is represented *directly.* (We could also use for example Mean Squared Error between the vector of outputs and the vector representation of the correct label, but it does not have this direct interpretation.) We will use θ as shorthand for the model parameters, in this case $\mathbf{W}, \mathbf{b}$; $\mathcal{D}$ denotes the data. The log-likelihood of the data is defined as:

$$\mathcal{L}(\theta, \mathcal{D}) = \sum_{i=0}^{|D|} log(P(Y = y^{(i)} \mid \mathbf{x}^{(i)}, \theta)) \tag{1.5}$$

where the expression $P(Y = y^{(i)} \mid \ldots)$ stands for the probability that the model assigns to the correct class for the i-th data item. (Given our choice of activation function, this is simply the activation of the $y^{(i)}$-th output neuron.)

To optimize the model parameters, we will now "simply" perform gradient descent on the loss function. The gradient for this loss function is well-known, but gradients for more complex networks can be hard to compute. For feedforward networks with certain classes of activation and loss functions, the *backpropagation* algorithm for computing gradients with respect to each parameter is used extensively since a famous 1986 paper [69].[6] The idea of backpropagation is that it is possible to find out how much each neuron and connection between neurons contributes to the error in the output layer and update the parameters accordingly by propagating the error backwards through the network, as though it were an input signal traveling from the output layer to the input.

For a long time, only networks with at most one hidden layer were being successfully used: although the power of multi-layer networks was hypothesized, effective training procedures were not known (or testable, given hardware at that time.)[7] Some reasons why training multi-layer networks by simple backpropagation with standard sigmoid units is difficult are described by Glorot and Bengio in [29]. Today, it is possible to train large and deep networks with back-propagation thanks to regularization techniques such as *dropout*, improved optimization algorithms and different activation functions (see 3 for an overview).

[6](Backpropagation was discovered earlier, in the 1960's, but this paper was seminal for its – well – propagation.)

[7]For a short history of neural networks including references to relevant publications, see the Wikipedia page on neural networks: `https://en.wikipedia.org/wiki/Artificial_neural_network#History`

For undirected networks, however, backpropagation is not applicable and different methods have to be used, such as *contrastive divergence.* (More on undirected networks will be said in 2.2.)

Fortunately, automatic differentiation software with focus on neural networks is available today which can eliminate this mathematical bottleneck; we have made extensive use of this functionality [8].

The strategy of choice for training neural networks by gradient descent is *mini-batch* gradient descent, a tradeoff between computing the gradient for the entire dataset (*batch* gradient descent) and for each item (*stochastic* gradient descent, SGD), as it combines the convergence properties of SGD with fast implementations of matrix operations (the input x in mini-batch gradient descent is not a single vector, as in SGD, but a matrix).[8]

Stochastic Gradient Descent is by far not the only way of training a neural network. Other optimization techniques can be applied, like the Levenberg-Marquardt method or conjugate gradients, and methods that utilize only a part of the information from the gradient have also been successful. (We will briefly describe this plethora of methods in the corresponding section of the Related Work chapter 3.3.

1.3 Other architectures

Aside from feedforward networks and undirected energy-based models (which we will discuss in section 2.2), two other architectures feature prominently among neural networks: *Recurrent Neural Networks* (RNNs) and the aforementioned *Convolutional Neural Networks.*

Recurent networks introduce directed edges that loop back into the layer they came from. This can be understood as a *history* and is naturally fit for sequential data – signal processing, like speech recognition, video processing or music information retrieval, or language modeling. By "unrolling" the recurrent network in time, we can recover a dynamic probabilistic graphical model similar in appearnace to a (rather complicated) Hidden Markov Model (HMM) [78]. The biological inspiration for RNNs comes from areas of the brain such as the Hippocampus or Lateral Genicular Nuclei, where some neurons from later parts of the processing pathways send signal to the areas passed previously.

Convolutional neural networks are inspired by the architecture of the retina. Several prevalent types of neurons participate in processing the raw "pixel-wise" (photoreceptive-cell-by-photoreceptive-cell) visual signal and significantly compacting it before passing it towards the lateral genicular nuclei and the visual cortex: cells that integrate signals from overlapping patches of the photoreceptive cells (*bipolar* cells) and other cells that discriminate between incoming signals through inhibiting their neighbors in the neuron layer (*horizontal* and *amacrine* cells). This action is simplified to two alternating types of layers: *convolutional* layers, which mimic integrating the signal of the regular overlapping patches that the horizontal cells process, and *max-pooling* layers, which inhibit some signals and let others pass uninhibited. Convolutional neural networks have become state-of-the-art in various image processing tasks, including the famous MNIST

[8]The minibatch variant is often implied in literature when the term SGD is used.

dataset [47].

2. Models in Deep Learning

In this chapter, we briefly introduce the models that are the cornerstones of representation learning in deep learning. These “building blocks” of deep architecutres are *Autoencoders* and *Restricted Boltzmann Machines*. (Much of the text in this chapter is based on the excellent tutorials provided by the LISA lab of the University of Montreal [50].)

2.1 Autoencoders

Autoencoders consist of three layers: visible inputs, hidden units and visible outputs. The visible layers both have d_v units, the hidden layer has d_h units.[1] The network is parametrized by two sets of weights and biases: encoding weights $\mathbf{W}$ and bias $\mathbf{b}$ and decoding weights $\mathbf{W}'$ and bias $\mathbf{b}'$. Typically, the weights will be *tied*, so that $\mathbf{W} = \mathbf{W}'^T$. The network uses *encoder* activation a and *decoder* activation a'. (In practice, the activation functions are often the same.)

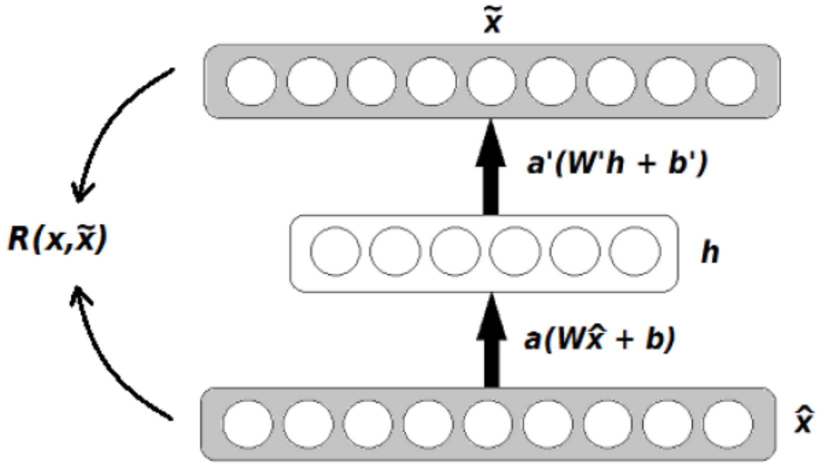


Figure 2.1: A plain autoencoder

The loss function is *reconstruction error*, which is a function of the input and its *reconstruction*. The reconstruction $\tilde{x}$ of an input x is defined as:

$$\tilde{x} = a'(\mathbf{W}'a(\mathbf{W}x + \mathbf{b}) + \mathbf{b}') \tag{2.1}$$

which is simply the result of computing the hidden layer activation h and then using the backward weights and bias to compute activation of the visible units. Autoencoders commonly use *tied weights*, where $\mathbf{W}' = \mathbf{W}^T$.

The loss function is then defined as

$$\mathcal{L}(x \mid \mathbf{W}, \mathbf{b}) = \mathcal{R}(x, \tilde{x}) \tag{2.2}$$

[1]We will use this notation for visible/hidden layer size throughout.

where $\mathcal{R}$ is the reconstruction error function. Common reconstruction errors are the Mean Squared Error

$$\mathcal{R}_{MSE}(x, \tilde{x}) = \frac{1}{N} \sum_{k=1}^{N} (x_k - \tilde{x}_k)^2 \tag{2.3}$$

or, for autoencoders with binary units, cross-entropy loss

$$\mathcal{R}_{XEnt}(x, \tilde{x}) = - \sum_{k=1}^{N} [x_k \; log \; \tilde{x}_k + (1 - x_k) \; log \; (1 - \tilde{x}_k)] \tag{2.4}$$

Cross-entropy can also be used for autoencoders with values interpreted as probabilities of the given unit turning on.

Autoencoders are thus encouraged to learn representations of the input that encode the input in d_h dimensions so that it is possible to reconstruct it as closely as possible – in other words, this means that the representations lose as little information about the input data as possible. (For a detailed analysis of how information flows in autoencoders, see [80].)

If $d_h < d_v$, interpreting the representation learned by the autoencoder is straightforward: it is a good (in the sense of the loss function) compression of the input data. The smaller amount of hidden units forces the autoencoder to discover some useful structure in the data that will allow it to represent it efficiently. (If the inputs were white noise, an autoencoder would not help us very much. However, if for example each even inputs were generated from low-variance Gaussians centered on the previous odd input, the autoencoder with $d_h = d_v/2$ could easily learn that the weights for inputs v_1, v_2 and v_3, v_4 etc. can be very close to each other.) Autoencoders have been used in this way for example to find compact representations for documents [35] or to compress the benchmark MNIST dataset down to 30 binary features [33]. The reconstructions obtained with autoencoders easily surpass linear methods such as PCA or LSA (see visualizations in [33]), although the learned representations have recently been matched by simpler methods on some classification tasks [15] [16] [59].[2]

The $d_h > d_v$ case is more interesting. While we are not forcing the autoencoder to find structure in the data by looking for a more compact representation, we can still coax it into learning representations with useful properties, such as sparsity [58] [51]. Also, simply having many non-linear projections of features can help us significantly for classification, as the resulting representation may make classes more linearly separable.

However, when there are more hidden than input/output neurons, one of the encodings that minimizes reconstruction error is for d_v neurons of the hidden layer to copy the input neurons and the rest will just always stay off. With $d_h >> d_v$, this solution may even be relatively sparse. However, such a representation does not tell us anything interesting; we want to avoid it. To this end, more complex autoencoders are designed: the *Denoising* and *Contractive* autoencoders.

[2]The results in [35] and [33] have been obtained by using *stacked* autoencoders, which we will describe in a later section 2.1.2.

2.1.1 Denoising autoencoders

Denoising autoencoders try to avoid learning the identity function by running the network on a noisy version of the data, but measuring reconstruction error against the orignal noise-free input. This forces the autoencoder to learn to de-noise the inputs (hence “de-noising”), making the identity an unworkable representation.

The reconstruction $\tilde{x}$ is obtained as

$$\tilde{x} = a'(\mathbf{W}'a(\mathbf{W}\hat{x} + \mathbf{b}) + \mathbf{b}') \tag{2.5}$$

where $\hat{x}$ is the *corrupted* (=noisy) version of the input. The reconstruction error is computed from the uncorrupted input as $\mathcal{R}(x, \tilde{x})$.

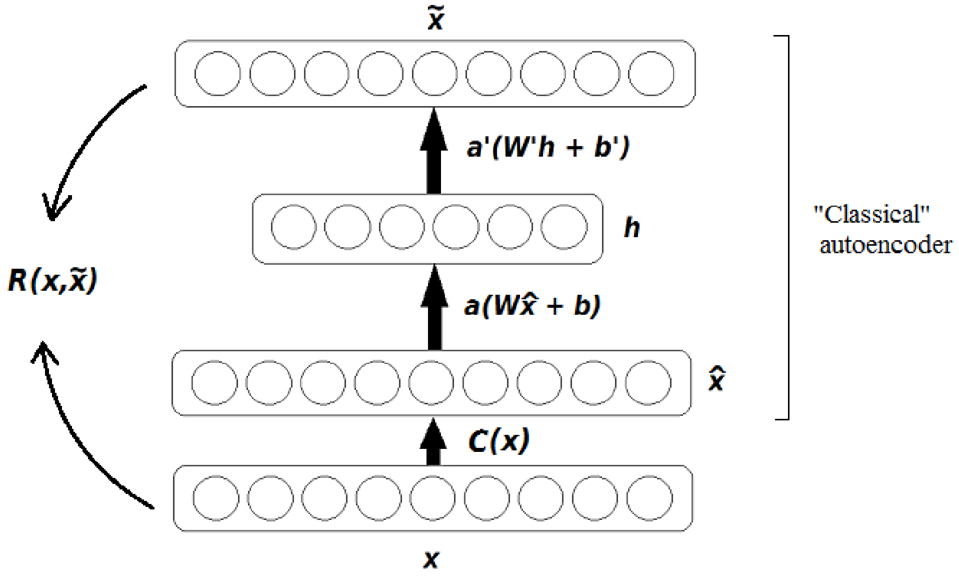


Figure 2.2: A denoising autoencoder. Note the relationship to the “plain” autoencoder and how the reconstruction is computed.

There are many possible corruption functions $\mathcal{C}$ that compute $\hat{x} = \mathcal{C}(x)$. The basic corruption function simply switches each input neuron off with a given probability. Noise can also of course be generated from an appropriate distribution (Gaussian noise) or *salt-and-pepper* noise can be used, which switches a proportion of inputs randomly to either minimum or maximum activation. A generalization of the denoising principle to multi-layer architectures and theoretical relationships between types of noise have been described by Poole et al. [62].

The relationship of denoising autoencoders to probabilistic models has recently been described in some detail by Bengio et al. [7].

2.1.2 Stacking autoencoders

Individual autoencoders can be stacked on top of each other just as any other neural network layers. The hidden layer becomes the input layer of the next autoencoder, etc. Stacking de-noising autoencoders this way with greedy pre-training of each layer separately has led to competitive performance on benchmark datasets [80].

2.2 Restricted Boltzmann Machines

A different approach is taken by generative undirected models called *Restricted Boltzmann Machines*, or RBMs. These models belong to the class of *energy-based* models, inspired by systems from statistical physics. We will first introduce the framework for energy-based models and then continue towards RBMs.

2.2.1 Energy-based models

Energy-based models assign an energy E to each state of the model's variables (in our case, neurons). Training the model means modifying the energy function – through changes in parameters – so that it has desirable properties: it should assign low energies to states that correspond to states that we think should happen, like samples from data, and high energies to states that should not happen, like all neurons set maximum activation. Through the energy function, a probability distribution of the data $\mathbf{x}$ is defined:

$$P(x) = \frac{e^{-E(x)}}{Z} \tag{2.6}$$

The normalizing factor Z is called the *partition function*, in line with the physical inspiration. It is defined as

$$Z = \sum_x e^{-E(x)} \tag{2.7}$$

Learning (like the linear regression example in the introduction 1.2) utilizes negative log-likelihood $\updownarrow$ of the data as the loss function:

$$\mathcal{L}(\theta, \mathcal{D}) = -\frac{1}{N} \sum_{x^{(i)} \in D} log\ p(x^{(i)}) \tag{2.8}$$

In this setup, x was composed of visible variables (input neurons) only. (Visible Boltzmann Machines are an example of such a model.) However, these models have limited use, as their expressive power is limited. The expressive power of energy-based models is greatly increased with the introduction of hidden variables, such as a hidden layer of neurons. The energy-based distribution becomes:

$$P(x) = \sum_h P(x, h) = \sum_h \frac{e^{-E(x,h)}}{Z}. \tag{2.9}$$

We will now define the physics-inspired notation of *free energy*, which will later allow us to write the data gradient in a fortunate form. The formula for free energy is:

$$\mathcal{F}(x) = -\,log \sum_{h} e^{-E(x,h)} \tag{2.10}$$

so that we get

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \qquad \text{with} \qquad Z = \sum_{\tilde{x}} e^{-\mathcal{F}(\tilde{x})} \tag{2.11}$$

(Note that $\tilde{x}$ goes over *all possible* points of the space in which the data is.) We can now express the gradient of the data negative log-likelihood in terms of this free energy (the detailed derivation is included as an attachment 9.2):

$$-\frac{\partial\, log\, p(x)}{\partial\theta} = \frac{\partial\mathcal{F}(x)}{\partial\theta} - \sum_{\tilde{x}} p(\tilde{x}) \frac{\partial\mathcal{F}(\tilde{x})}{\partial\theta} \tag{2.12}$$

The terms of the gradient are called the *positive* and *negative* phase, based on their effect on the density $P(x)$. (For a detailed explanation, see Hinton's original paper on contrastive divergence [36].)

The second term is an expectation of the free energy gradient over the model distribution. This is intractable, because it would require summing over the entire space of possible $\tilde{x}$s (configurations of input neurons), which even with just binary units is exponentially large. However, it can be approximated by taking a set of sample configurations $\mathcal{N}$:

$$-\frac{\partial\, log\, p(x)}{\partial\theta} \approx \frac{\partial\mathcal{F}(x)}{\partial\theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{x}\in\mathcal{N}} \frac{\partial\mathcal{F}(\tilde{x})}{\partial\theta} \tag{2.13}$$

The samples from the model distribution are called *negative particles.* They represent a "fantasy" of the model: if we left it alone, it would be spontaneously generating particles like this. However, we want it to model the *data* – so the gradient points from what the model is currently dreaming towards where the data lives. We are thus gradually convincing the model to dream about the data.

Equation 2.13 would be all we need to build a training algorithm for energy-based models – if we could sample from $p(\tilde{x})$. This, of course, can be done (for some models easier than for others). We will now shortly interrupt this line of thought to define Restricted Boltzmann Machines and then come back to show how the equation 2.13 is applied to RBMs.

2.2.2 The Restricted Boltzmann Machine model

A Restricted Boltzmann Machine consists of two layers: an input layer and a hidden layer, with each hidden neuron connected to each visible neuron. As opposed to the Autoencoders and feedforward networks, RBMs are *undirected*, meaning that signal can flow both ways. (This also means backpropagation is unapplicable.) A Restricted Boltzmann Machine looks like this:

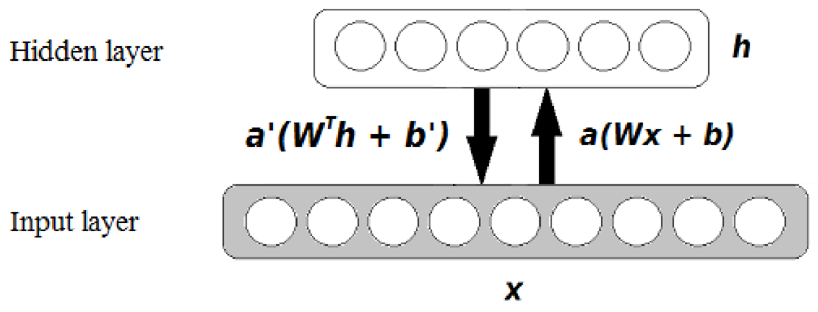


Figure 2.3: A visualization of a RBM. The equations show how to compute the activation of the hidden layer given the visible layer and vice versa.

The neurons of an RBM are commonly restricted to be binary (either 0 or 1): they are either on, or off. The activation of a neuron is interpreted as the probability of the neuron being on. Because of how the model is structured, neurons are conditionally independent given the opposit layer:[3]

$$p(x_i = 1 \mid h) = \prod_j p(x_i \mid h_j) \tag{2.14}$$

$$p(h_j = 1 \mid v) = \prod_i p(h_j \mid x_i) \tag{2.15}$$

This formula can be generalized for non-binary RBMs, such as RBMs with a real-valued Gaussian visible layer [33] [82].

The energy function for a RBM is defined as:

$$E(x, h) = -b'x - bh - h^T Wx \tag{2.16}$$

2.2.3 Contrastive Divergence

Now that we have defined Restricted Boltzmann Machines, we return to the equation 2.13, the gradient of the negative log likelihood of energy-based models. In order to train a RBM, we need to sample the *negative particles* $\mathcal{N}$ from the model.

Given the independence structure of RBMs, this can be easily done using alternating block Gibbs sampling: first sample the hidden layer, then the visible given the hidden, etc. This is a standard Markov Chain Monte-Carlo approach. However, obtaining an unbiased sample from the Markov chain can take many iterations, making learning too slow, and there is no reliable way of telling when the chain has reached equilibrium. Furthermore, the estimated gradient has a large variance [14].

This is where the *Contrastive Divergence* (CD) algorithm of Hinton [36] has proven a breakthrough. Instead of directly optimizing the negative log-likelihood,

[3]Recall the rules for conditional independence in probabilistic graphical models.

it optimizes the difference of two Kullback-Leiber divergences. We will introduce some more notation:

- P_0 is the *data* distribution (the "0" in the notation stands for beginning a Markov chain there)
- P_∞ is the *model* distribution, which is what we get if we actually run the Markov chain to equilibrum
- P_n is the model distribution we get after running the Markov chain for n steps, starting at P_0.

Optimizing negative log-likelihood is minimizing the KL-divergence $KL(P_0 \| P_\infty)$. In contrast, CD optimizes the difference between two KL-divergences [36]:

$$KL(P_0 \| P_\infty) - KL(P_n \| P_\infty) \tag{2.17}$$

This cancels out the expectation over the equilibrum P_∞. The Contrastive Divergence algorithm then becomes at each point x_i of the data:

- Initialize chain at x_i in the visible layer,
- Run alternating block Gibbs sampling for K steps, where K is most often 1,
- Take the last visible sample as a negative particle from eq. 2.13, plug into formula and compute gradient

Of course, the algorithm can be easily parallelized so that the Gibbs sampling is run for multiple x_i in one matrix operation.

The first key observation of Contrastive Divergence is that while for obtaining an unbiased sample we need a long, well-mixed chain, for *training* the model, a biased sample of just a few steps may suffice; in fact sampling the hidden and visible layer just once is enough. The second key observation is that we can initialize the chain with the data point we're currently processing, and that it may in fact be *better* for learning from the given data point, in combination with the very short chain: the one step of CD shows the direction in which the Markov chain has a tendency to wander off, thus correcting for it.

As Hinton writes [36]:

"Another way of understanding contrastive divergence learning is to view it as a method of eliminating all the ways in which the PoE model would like to distort the true data. This is done by ensuring that, on average, the reconstruction is no more probable under the PoE model than the original data vector." [36]

A good introduction to the mathematics of RBM training has also been written written by Fischer and Igel [27]. The theoretical properties of CD have been described in [14] and more recently in [5].

A problem with CD-K is that the Markov chain may not mix fast enough near x_i. This is addressed by the *persistent contrastive divergence* (PCD) algorithm [79]. The central idea of PCD is that instead of starting a new chain with each data point, we keep several chains running throughout training and sample negative particles from them.

2.2.4 Variants of RBMs

Given the probabilistic nature of RBMs, multiple interesting priors can be applied, such as the Spike and Slab prior to encourage a "spiking" type of sparsity instead of only encouraging feature means to stay low.

A notable variant of RBMs for modeling documents as bags-of-words is the Replicated Softmax model of Salakhutdinov and Hinton [70] in which the hidden (forward) activation is a standard sigmoid, but the visible (backward) activation is the softmax function, so that the vector $p(\mathbf{x}' \mid \mathbf{h})$ forms a categorical distribution. The input neurons each represent one word in the dictionary (typically a couple of 1000's of most frequent words in the data). To recover a negative particle for a document x of D words, this categorical distribution is sampled from D times. The hidden bias units have to be scaled by D in the energy function as well; this can be understood as sampling D times from the empirical categorical distribution over the input document's word frequencies. (Note that the article [70] refers to the categorical distribution as multinomial.) The scaling by D allows Replicated Softmax to consistently model bag-of-words documents of varying lengths.

Multi-layer RBMs called *Deep Boltzmann Machines* (DBMs) can be created by stacking RBM layers on top of each other, just like when building autoencoders. DBMs have been very successful in representation learning and applications. An extensive review has been written by Bengio [4].

3. Related Work

The field of deep learning is in an explosion of activity since 2006. Although relatively young, it has accumulated an impressive track record, pushing state of the art on many tasks by a very significant amount. This has been most notable in image processing, where Convolutional Neural Networks represent the new state of the art on content-based tasks, especially on the ImageNet-1000 classification challenge [45] and PASCAL-VOC datasets.[1] Considerable headway has also been made in natural language processing, some of which will be reviewed in 3.2.

3.1 Deep Learning

Multi-layer, or *deep*, neural networks have been proposed as soon as the idea of mimicking the biological neural system itself. However, until recently (2006), multi-layer neural networks performed poorly. (An analysis of some of the reasons can be found in [29].)

Importantly, the raw computing power to properly train large deep networks simply wasn't available to researchers. This has changed during the last decade, most notably with the advent of massive parallelism in general-purpose GPU computing, for which the matrix multiplication algorithms of neural network signal propagation are very well-suited. Most large neural networks today are trained either using GPUs [45], or a distributed system [54], or both.

The other important development are successes of recent methods of training multi-layer models, most notably *unsupervised pre-training.*

3.1.1 Unsupervised pre-training

A significant development was the discovery of efficient inference for certain classes of neural networks and the *layer-wise unsupervised pre-training* methodology proposed by Hinton and Salakhutdinov [33]. Layer-wise pretraining helps combat the vanishing gradient problem[2] in deep networks by initializing the network to the areas in the parameter space in which the data actually reside [6], so that no large gradient updates during supervised training (called *fine-tuning* in this context) are necessary. This helps the network generalize better, because it ensures that it is actually modeling a distribution where the data come from [24]. Unsupervised pre-training is most useful in situations where there is a large amount of unlabeled and only small amount of labeled data available; thanks to modern techniques to combat overfitting and other problems of training feedforward networks, it is now possible to train large and deep supervised neural networks using

[1]`http://pascallin.ecs.soton.ac.uk/challenges/VOC`

[2]The error from output neuron j propagates over asymptotically more and more connections, thereby updating connections in the layers nearer to the input with a rapidly decreasing magnitude, easily canceling out with other such dilluted, "fizzled-out" gradients. In the first step of backpropagation, the gradient from j propagates over as many connections as there are neurons in the previous layer. However, from the previous layer to the layer before that, the error influences the connection between *every pair* of neurons - proportionally to how much it influenced the given backpropagating neuron in the previous iteration.

plain backpropagation.[3]

3.2 Deep Learning in Natural Language Processing

Probably the most notable task in Natural Language Processing where deep learning methods have made a recent impact is Language Modeling. The first oft-cited Neural Network Language Model (NNLM) is the work of Bengio et al. [2]. Building on this model, Mikolov et al. [54] has made remarkable headway in learning a representation in which meaningful operations on words can be performed using arithmetic operations on the word representations (such as $king - man + woman = queen$, or $\frac{Germany}{Berlin} = \frac{France}{Paris}$). Recurrent neural network language models, also by Mikolov [55], have been found to perform excellently using the representations obtained from skip-grams [54].

Also interesting is the SENNA[4] system of Collobert, Weston et al. [18], which shares representations across multiple tasks: language modelling, part-of-speech tagging, semantic role labeling, syntactic parsing or named entity recognition.

Models for entire documents based on the bag-of-words approach have also been explored. One such model is the Replicated Softmax model by Salakhutdinov and Hinton [70], which was also used in [73]. On top of Replicated Softmax, the Over-replicated Softmax model by the same research group[5] has been published in 2013 [75]. A thorough description of various previous (non-deep learning) document models and their relationships, together with a Deep Boltzmann Machine for representing documents in a low-dimensional space, is given by the same authors in [35]. However, document modeling on word observations is very much an open problem [21].

3.3 Neural Network Training

Successfully training neural networks is not easy and may require a lot of parameter tuning. A substantial amount of work dedicated to training a good neural network therefore doesn't make it into published papers. In this respect, a remarkable resource are *practical guides* to training networks, such as LeCun's Efficient BackProp [48], Hinton's Practical Guide to training RBMs [38] or Bengio's Practical recommendations [3]. Another such paper is the guide to SGD by

[3]The deep learning mantra for supervised learning today is mostly ReLU + dropout + a lot of parameter tuning, not necessarily pre-training, as for instance one of the foremost deep learning researchers Yann LeCun noted in a recent interview on the Reddit website: `http://www.reddit.com/r/MachineLearning/comments/25lnbt/ama_yann_lecun` On a tangential note, this is a general feature of information on deep learning: a significant amount of expertise is available in condensed form from unofficial sources online, such as the `stackoverflow.com` Q&A and other stackexchange network site (stats), `metaoptimize.com` and the /r/machinelearning subreddit. While not the place for scientifically rigorous results, these sources are invaluable for orienting oneself within the field.

[4]Downloadable from `http://ml.nec-labs.com/senna/`

[5]More or less all recent advances in deep learning methods (not necessarily applications) have been brought about by three or four research groups: those of Geoffrey E. Hinton, Yoshua Bengio and Yann LeCun.

Bottou [10].

Many advances have also been made in neural network training methodology. Speedups and performance improvements have been made through new activation functions, optimization algorithms and other ”tricks” such as dropout or centering. We will shortly describe some of these techniques.

3.3.1 Optimization algorithms

While simplest, plain SGD backpropagation (or mini-batch SGD) is not always the best choice of optimization algorithm, especially as far as convergence speed is concerned. The two most important problematic aspects of SGD are the dependence of the step size on the magnitude of the gradient and the difficulty in choosing a learning rate. When the activation function has a sigmoid shape, the neuron can enter the *saturated regions* of the sigmoid where the magnitude of the gradient is very small and therefore SGD step size greatly decreases. This can be especially problematic in networks where sparse activation patterns are preferred, which encourage neurons to be either firmly off or firmly on and thus can force neurons into these regions of saturation. The problem of choosing a learning rate is related to this: if we choose a high learning rate to help the network parameters move around more and get un-stuck from local minima easier, we risk settling into an oscillating pattern; if we choose a learning rate that is too low, we might not get to the are around the optimum at all, unable to escape from a local minimum with saturated neurons. Due to these problems with “raw” SGD, a number of more advanced approaches has been successfully developed.

A straightforward strategy is to adapt the parameter update to the status of the training. One simple and successful [35] [38] method is *momentum*: when computing the parameter update, we remember the update from last time and add a certain percentage of it to the new update.

$$\Delta\theta_i = lr * \nabla\theta + m * \Delta\theta_{i-1}$$

(lr is the learning rate and m is the momentum; both numbers are between 0 and 1.) This helps SGD two-fold. First, it makes learning less sensitive to differences and noise between mini-batches, as it encourages the update to follow a more general trend. Second, it takes into account the update history: if consecutive updates point in generally the same direction, the momentum term will push the updates along faster. If a change of direction is needed, the update first needs to “brake”, which – while it may cause it to overshoot for example a turn in a loss function ravine – makes it resistant to smaller “bumps” on the way. However, once the learning starts to circle around a large enough optimum, the frequent changes in gradient direction will tone the momentum term down so that it can find the actual optimum.

Another algorithm that manipulates learning rate is *Adagrad* [23] and more recently *ADADELTA* [84]. While momentum adapts the direction of the update to recent learning history, these algorithms directly manipulate the learning rates to decrease component-wise with training time to reflect how close each parameter is to convergence. Adagrad has proved successful in training neural network language models on huge data in a distributed computing environment [54].

A third succesful alternative is the family of *resillient backpropagation*, rprop [66] [67] [65]. Instead of basing the step size on the gradient magnitude, it only uses the *sign* of the gradient. If the sign changes, rprop flips the direction of the update; the magnitude of the parameter update depends solely on how many updates have there previously been without the sign of the gradient of that particular parameter flipping (the un-flipped updates are incremented by a small constant term in each step, so that subsequent updates in the same direction go faster). In this respect, it is like a very rough approximation of the momentum, but without the $\nabla\theta$ term, completely bypassing the saturated gradient problem. The rprop family has proven surprisingly successful, both in terms of the optima found and learning speed [42] [39].

Yet another attempt to solve the SGD update step size problem has been made by Schaul et al. [71] who try to find an *optimal* learning rate based on the Hessian.

This brings us to the category of *second-order* optimization algorithms. The algorithms we have seen so far were based on the first derivative of the loss function; second-order methods take into account not just the slope, but also the curvature of the loss function surface and move the parameters accordingly. Because repeated computation of the Hessian is impractical for even medium-size neural networks (a couple thousand units), since it is quadratic in the number of parameters, which by itself is quadratic with the number of neurons, a *Hessian-free* approach has been suggested by Martens and Sutskever [52] [53]. The first paper includes a discussion of pathological loss function curvature, the second deals in more detail with convergence properties and practical settings.

The general second-order Levenberg-Marquardt method [49] can also be used, which is a curve-fitting method based on optimizing the least-squares criterion. This method approximates the Hessian by taking the outer product of Jacobians $J^T J$. (If this approximation has a significant error, convergence will be slower.)

A third popular second-order choice is the *Conjugate Gradient* (CG) method, which adds a line search component along the gradient direction to determine the update step size. Kostopoulos and Grapsa have written in 2009 a brief overview of standard CG methods and proposed a self-scaling variant in [44].

However, it should be noted that plain SGD and variants can and do outperform second-order methods on non-convex problems [11].[6]

For training Restricted Boltzmann Machines, an alternative to Contrastive Divergence has been proposed, *parallel tempering* [22]. This algorithm addresses the potentially slow mixing rate and stability problems of the Markov chain in Persistent Contrastive Divergence.

3.3.2 Other tricks

Several other training techniques have been recently developed and recommendations for various normalization and other preprocessing steps have been published in practical guides. The most influential technique is *dropout* [74]. Dropout is a powerful regularization technique that combats overfitting by randomly switching off a significant portion of hidden neurons (often half of them). This serves to reduce co-adaptation patterns between neurons, as they cannot rely on a partner

[6] `http://leon.bottou.org/projects/sgd`

being activated [34]. Dropout can also be seen as a generalization of the Denoising Autoencoder principle of evaluating the network's performance with artificially noisy inputs evaluated on noise-free targets. It has been very successful, especially for supervised training [45], and has become somewhat of a recipe.

3.4 Text-Image Models

The multimodal Deep Boltzmann Machine of Srivastava and Salkhutdinov [73] after which we fashion our model architecture, trains a stack of RBMs for each modality (text and images) separately and adds a joint representation layer on top of the individual modality stacks. Their dataset cosists of images and texts with 1-0 labels for each one of However, the text data used by [73] only contains keywords, not entire documents. They represent images by 3857-dimensional features, that were extracted by concatenating Pyramid Histogram of Words (PHOW) features, Gist and MPEG-7 descriptors (EHD, HTD, CSD, CLD, SCD). Aside from results on discriminative image category classification tasks, for generating images for keywords, they report a MAP of 0.614. However, they use a very permissive method of evaluation, where if a retrieved image overlaps in at least one category with the categories of the query image, it is considered relevant.

An approach that does not utilize deep learning is by Feng & Lapata [26]. In this work, image features are SIFT descriptors (histograms of edge directions), quantized using the K-means clustering algorithm into *visual words*. This is similar to the PHOW features of [73]. A standard topic model, Latent Dirichlet Allocation [9], is then trained jointly above both the visual words and text words. They reach a top-1 accuracy of 57.3 %.

Other text-image models include multimodal log-bilinear language models of Kiros et al. [43] that combine the language modeling approach with multimodal representation or an interesting multimodal experiment done by Socher et al. in [72]: using a recurrent neural network to capture recursive structure common across images and sentences.

4. Dataset

For our experiments, we have gathered a collection of journalistic articles and the accompanying images from 13 Czech news portals. Three of those are internet editions of tabloid magazines. We call the dataset *web-pic*.[1]

By far not all of the texts are high-quality journalism (even after excluding tabloids). Nevertheless, the composition of the dataset does reflect, to a degree, the publishing volume of large Czech journalistic portals.

Many articles have more than one illustrative image. However, some of these images may be connected only to specific portions of the texts. Determining and annotating this relationship in the dataset is reserved for future work.

It is necessary to note that this dataset is sparse with respect to recall: while we do have appropriate images for each text, we by far do not have *all* appropriate images for the given text, or at least a decent sample thereof; furthermore, we do not know *how* sparse the dataset is in this respect. The manual annotation described in 4.4 may go some way towards estimating the magnitude of this problem. (This sparsity is in direct contrast to the data of Srivastava and Salakhutdinov [**?**] who can use a rich "network" of connections between individual texts and images by exploiting the category labels of each of their data points.)

4.1 Statistics

The document counts and lengths for individual sources (subcorpora) are:

Subcorpus	# of docs	# of images
aha (T)	5418	7521
ble (T)	5473	15555
cen	2446	5495
den	763	1177
e15	519	910
eur	157	163
idn	16017	31462
kaf	276	798
nov	9036	15613
ref	660	809
sup (T)	2741	16638
tis	683	744
tyd	387	1176

Table 4.1: Subcorpora statistics

We only use articles and texts that are *not* reports of specific events. Illustative images for reporting articles are usually custom-made for the event that is being reported on, so there is no reason to automatically provide one; furthermore, the features that make a good accompanying image for such a report are

[1]The dataset will be available under this name from the LINDAT/CLARIN infrastructure. The image *features* (see 5.1 for details) will be released; the release of the images themselves is complicated legally and we are investigating our options.

very different from those that make a good illustrative image for e.g. a lifestyle article.

The dataset has been lemmatized and tagged using the MorphoDiTa morphological tagger [77]. Tags are positional; the tagset is described by Hajič in [32].

4.2 Dataset cleaning

Even after crawling and retrieving raw text and images, the dataset had to be further cleaned because of irrelevant images stemming from news outlet habits: embedding advertisement images into the texts, referencing other articles with images and other "junk" images randomly gleaned from the surroundings of the text. These spurious images were filtered out using simple rules hand-crafted for each subcorpus based on manual exploration of the roles of images of different sizes.

However, the dataset is most probably not completely clear in this respect; it is plausible that some junk images were missed during cleaning. This is a possible source of noise and the only estimate we have of its magnitude is the proportion of original images tagged by annotators (see 4.4.4).

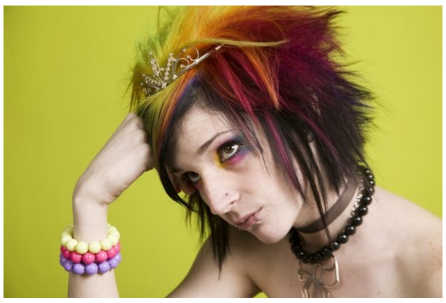
Further, we determined by manual inspection that the majority of the tabloid articles are accompanied by irrelevant images that serve merely to draw reader attention, or are pictures of specific persons for celebrity gossip articles. The proportion of such images was about 80 - 90 % in the *aha* and *ble* subcorpora and 50 % in the *sup* subcorpus. **In our experiments, we decided to discard the tabloid data.**

4.3 Text-image relationships

Usually, the images are assigned to the texts they illustrate not by the author of the text, but by an editor who specializes in choosing illustrative images (we will call him/her the *illustrator*). The goals of this illustrator are *not* to simply show what is in the article, but to *draw such readers to the text that will spend time with viewing the article.* This strategy results in a range of ways in which the image is relevant (or, in some cases, irrelevant) to the content of the text.

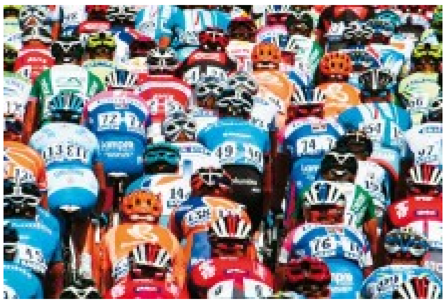
The prototypical case is a simple fit between the article topic and the content of the image (drawing readers that are simply interested in the topic). [2] In some articles, the illustrator chooses a tangential topic. Sometimes, there is only a very loose association between the content of the article and its accompanying image.

[2]The translation of the Czech texts from the examples to English was done by the author of this work.



To beat, or not to beat? On dealing with a teen who looks a little different. Earrings through the tongue, blue hair, a black cloak all the way to the ground or trousers eight sizes too large, with the crotch at the knees? If you are ashamed to go out in the street with your almost-adult progeny, then that's all right

Figure 4.1: Direct illustration of the topic of the article.



DANGEROUS MEDICINE What do the actress Jiřina Bohdalová and top athletes have in common? Ozone in their blood. They are getting high on their own blood infused with a mixture of oxygen and ozone. So do those who want to slow down aging and cure a variety of diseases associated with it. However, this medicine is dangerous and a felony.

Figure 4.2: Tangential topic illustration

4.4 Annotation

The downloaded web-pic dataset is *sparse*, in the sense that it exhibits poor recall – only one or sometimes a few of many possible illustrative images have been chosen for each article. Therefore, we have performed further manual annotation.

Annotators were shown *items* consisting of a text and twelve images. They were asked to tag appropriate and also inappropriate images. Appropriate images were defined as ”images that help the gist or message of the article get across”, inappropriate images were defined as ”those that would lead the reader to misunderstand the gist/message of the article”, in the sense that if they were used, they would promote an idea of what the article was trying to say incongruent with the actual message. (For example, an article titled ”The beauty of the Azores” that talks about how great the islands are for a slow family holiday should not be accompanied by a photo of extreme sports.) The concept of inappropriateness is based upon the fact that while an image can be related to the topic of the article, associating that image with the article would produce a discrepancy for the reader: based on the image, the reader would expect a different message than the article is trying to convey. This is a situation that the image retrieval system could easily get into – while it retrieves an image related to what is being talked about in the article, the image is still wrong.

However, while appropriateness of illustrative images is easily understood in intuitive terms, inappropriateness was an experiment with a very uncertain outcome: it is not clear whether such a category can be meaningfully – with multiple annotators agreeing – annotated.

Admittedly, the appropriateness and even more inappropriateness of an image cannot be quantified well and described exactly. We rely on redundant annotation items for an estimate of how reliably these concepts can be annotated. This is one of the key questions of our annotation: *how well can people agree on what constitutes a good (bad) illustrative photo?*

More accurately, the main goals of the annotation were:

- To reduce dataset sparsity for evaluation of the image retrieval model
- To estimate how severe the sparsity problem is
- To measure how robust the concept of an appropriate illustrative image is
- To gain a high-quality dataset for further experimentation, or to exclude the possibility of getting one

4.4.1 Inappropriate images

The concept of inappropriate images merits further discussion. Appropriateness is not unidimensional: while the *content* of an image is a category that plays a major role in whether the image illustrates a text well, so does the *effect* it has on the reader.

Problems with inappropriate items were pervasive. A percentage of the annotated articles had specific topics like a species of insect or a health problem; images that were visually very similar to the topic – some other insect, a different kind of rash, etc. – *were* technically inappropriate and misleading, by associating the wrong "visual fact" with the topic of the text, but not in the way the category was intended to function. That this is a problematic situation was understood by the annotators as well, as pointed out frequently in the collected feedback.

4.4.2 Annotation process

The annotation was divided into two rounds, each round taking under a week to complete. The first round was aimed at finding common problems, mostly misinterpretation of instructions, and getting the first rough estimates on how many images will be tagged, what agreement to expect, etc. The second round was aimed at producing an already viable dataset which will be subject to further manual examination: reconstructing *why* the annotators chose to tag the images they tagged, what are some reasons for disagreement, etc. This qualitative evaluation is yet to be undertaken.

Annotators were given incentives both for quantity and for quality. Quality was based on agreement: annotators were rewarded according to whether they were able to tag the same images as other annotators. Since the number of tagged images itself was the quantitative incentive, we chose precision ($P(tagged \mid taggedbyother)$) as the metric to evaluate annotator quality. Together with sharing detailed information on agreement and item counts after the first round, this balance was sufficient to keep the average number of images per item consistent between annotators.

4.4.3 Annotation item selection

Annotation items consisted of one text and 12 proposed images. Of these, some were picked as the closest to the original image associated with the text, the rest were chosen randomly from the rest. We tried several settings for the balance of closest/random images; the most satisfactory balance between obtaining results for a text and exposing the annotators to images visually unrelated to the original image was struck at 7 closest and 5 random images. (This ratio was selected by manually inspecting the generated annotation items.)

With a probability of 0.5, the original image was left in the annotation item. The ability of annotators to recover these original images serves as an important measure of the agreement we can expect on this task. By leaving some out, on the other hand, we try to avoid situations where one image is obviously much better than all others, thereby decreasing the willingness of annotators to tag images that while perhaps not as appropriate could work anyway.

4.4.4 Quantitative evaluation

The reported results are from the second round of annotations, as the first round served mostly as fine-tuning for annotation instructions and was plagued by several systematic problems that during the second round were fixed (misreadings of the definition of inappropriateness, issues with timing, etc.). In total, 13 annotators participated in the second round (some, of course, more so than others).

Annot.	# Items	Avg./Item	# App. Im.	Avg. A.	# Inapp.	Avg. In.
Total:	5212	2.625	8462	1.624	5217	1.001
mar	239	4.038	627	2.623	338	1.414
ter	239	1.971	326	1.364	145	0.607
kri	1090	2.290	1507	1.383	989	0.907
cic	1034	2.591	1723	1.666	956	0.925
mag	259	2.378	444	1.714	172	0.664
jir	54	3.407	87	1.611	97	1.796
dag	124	2.460	305	2.460	0	0.000
ann	128	2.367	197	1.539	106	0.828
jan	226	2.730	441	1.951	176	0.779
haj	7	2.857	13	1.857	7	1.000
kub	323	2.065	414	1.282	253	0.783
ala	809	2.698	1174	1.451	1009	1.247
ven	680	3.196	1204	1.771	969	1.425

Table 4.2: Total annotated items and images for each category and annotator. The columns are: annotator name, no. of items processed, average no. of images tagged per item, total no. of tagged appropriate images, average no. of appropriate images per item, total no. and average of inappropriate.

As the basic means of measuring inter- and intra-annotator agreement, we measure *f1-score*. F1-score, also known simply as f-score, is the harmonic mean of *precision* and *recall*. Precision for a category C (e.g. appropriate) for annotator A_1 with respect to annotator A_2 in item i is defined as the proportion of images

Annot.	F-score	% of median F-score	# of dupl. items
Total	0.603		2030
mar	0.593	0.998	199
ter	0.624	1.049	182
kri	0.602	1.013	835
cic	0.649	1.091	804
mag	0.595	1.000	205
jir	0.589	0.990	38
dag	0.587	0.988	97
ann	0.545	0.916	101
jan	0.606	1.019	167
haj	0.533	0.897	5
kub	0.613	1.031	256
ala	0.592	0.996	663
ven	0.555	0.933	508

Table 4.3: Average F-score on appropriate images per annotator.

tagged as C by both A_1 and A_2 (referred to as "hits") in the set of images tagged as C by A_1.

Recall is then the proportion of hits in i in the set tagged as C by A_2.

Recall indicates whether annotator A_1 agreed with A_2 on which items to tag, precision indicates whether A_1 agreed with A_2 on which items *not* to tag. Low precision and high recall means that while the annotators agreed on some of the images, A_1 had much more relaxed criteria for the category. And vice versa: high precision and low recall means that A_1 was stricter than A_2. The worst case is, naturally, both low precision and low recall, which means that the annotators disagree on what constitutes an appropriate or inappropriate image for the given text.

We also measured how often the annotators identified the original image associated with the article (see tab. 4.5) when it was present in the annotation item.

Annot.	F-score	% of median F-score	# of dupl. items
Total	0.446		2030
mar	0.087	0.190	199
ter	0.459	1.000	182
kri	0.520	1.133	835
cic	0.482	1.050	804
mag	0.481	1.047	205
jir	0.379	0.827	38
dag	0.505	1.101	97
ann	0.418	0.911	101
jan	0.405	0.882	167
haj	0.400	0.872	5
kub	0.556	1.212	256
ala	0.445	0.969	663
ven	0.346	0.754	508

Table 4.4: Average F-score on inappropriate images per annotator.

Annot.	% orig. as app.	# as app.	% inapp.	# inapp.	% untagged	# untg.
Total	0.792	2028				
mar	0.922	106	0.000	0	0.078	9
ter	0.781	89	0.053	6	0.167	19
kri	0.751	411	0.062	34	0.186	102
cic	0.828	404	0.043	21	0.129	63
mag	0.764	97	0.094	12	0.142	18
jir	0.724	21	0.069	2	0.207	6
dag	0.864	51	0.000	0	0.136	8
ann	0.778	42	0.000	0	0.222	12
jan	0.885	100	0.009	1	0.106	12
haj	0.800	4	0.000	0	0.200	1
kub	0.819	122	0.047	7	0.134	20
ala	0.740	304	0.063	26	0.197	81
ven	0.769	277	0.097	35	0.133	48

Table 4.5: Proportion and number of original illustrative images tagged by annotators as appropriate, inappropriate or left untagged

5. Preprocessing

In order to make the text and image features usable for training a deep architecture over them, several preprocessing steps have to be applied, both for text and images.

5.1 Image Processing

For image processing, we use the state-of-the-art convolutional network of Krizhevsky et al. [45] (referred to as ImageNet CNN in this chapter) to provide the image features, using the Caffe [40] implementation. We use the output of the last 4096-neuron layer on the images in our dataset as the image features, and "re-purpose" them for our tasks. (The same "trick" was used for example by Girshick et al. in [28] for adapting the ImageNet *classification* to object *detection*.)

The network's output 1000-neuron layer is not used because it encodes predetermined categories that do not correspond well to topics found in our dataset (ImageNet topics are for instance "Golden Retriever" or "castle", the topics found in the news texts are rather "banking", "health", "crime" or "holidays". Notably, the ImageNet topics do not have a category for people, while the news texts are almost exclusively concerned with human-centric topics.) While it may be possible to express those topics in terms of the 1000 ImageNet categories, we chose to use the representation that should be better-suited to learn *various* sets of categories and more open to re-purposing.

5.1.1 ImageNet CNN

In order to correctly process these features, we need to understand both what the features represent and how these features were obtained. The network has eight layers, five convolutional and three fully connected. The convolutional layers are a black box to us, as all their "work" is processed by the first fully connected layer. Because the network was trained by standard backpropagation, we cannot discount the influence of the last layer, because that is where gradients backpropagated from. However, the features themselves were produced as a function of the first fully connected layer activation and our layer's weights and bias; this is what we will focus on.

Units of the previous layer (and of all layers in the ImageNet CNN) are *Rectified Linear Units* (ReLU). They use the activation function $f(x) = max(0, x)$, as described in [45]. ReLUs have since been found useful on other architectures than CNNs, notably also Restricted Boltzmann Machines [37]. ReLUs are non-saturating units that can output any value in the range $< 0, +inf >$. (In addition, in the ImageNet network, the outputs of ReLUs in the convolutional layers are further transformed to promote competition between neurons for large activations.) This has works very well in supervised classification tasks, where the target values of $0, 1$ in the output layer act as a regularization on the absolute values of activations throughout the network. However, in an unsupervised setting, this regularization factor has to be supplied from somewhere else. Therefore, in order to learn RBMs or Autoencoders above these image features, we need to devise a suitable prior or a scaling scheme to model these inputs.

5.1.2 Image data properties

We first visualize a sample of the dataset:

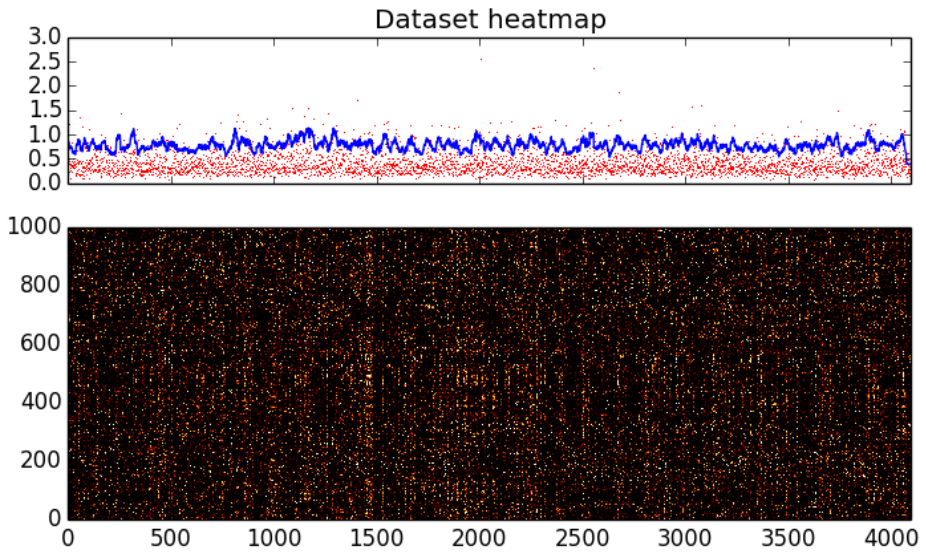


Figure 5.1: A heatmap of the ImageNet layer activations, with 1000 randomly chosen images. Lighter colors correspond to higher activation (white represents an activation of 4.00 or higher). The top plot shows average activation of each neuron in the sample. (The blue line represents a moving average over 20 adjacent neurons and is scaled up by a factor of 2, to help visualize variations in average activation. The peaks correspond to lighter columns of the heatmap.)

The overall sparsity (proportion of zeros) of the image dataset is about 61.4 %: slightly more than one in three activations is non-zero. The average correlation between feature pairs on this sample is slightly under 0.005, showing that co-adaptation in this layer has been combatted successfully.

However, the dataset doesn't lend itself easily to interpretation – we cannot exactly say what the activity of each neuron means by itself. (This is a general problem with neural networks: they are often difficult to interpret.)

The applicability of centering to zero mean is contrary to the intuition that the data vector are *sparse*, with many of the neurons set to 0. This is especially true for the text data, which is notoriously sparse: our text data has 0.354 % of nonzero input activations after filtering. Centering, in this context, would move all the zero activations to slightly *less* than zero, losing us the property of neurons simply not contributing.

In order to compensate for the apparent non-uniform mean activation of the image features, we applied the covariance-based scaling proposed by LeCunn et al. in [48], eq. 16. They propose scaling each feature so that the covariance C_i of each feature f_i is "about the same", with C_i defined as

$$C_i = \frac{1}{N} \sum_{n=1}^{N} (x_n^{(i)})^2 \tag{5.1}$$

where $x_n^{(i)}$ is the value of the i-th feature in data point x_n. To scale the features, we need to compute the covariance scaling coefficient c_i. We set the target covariance $\bar{C}$ as the mean of all covariances C_i. The coefficient is then obtained by solving for c_i:

$$\frac{1}{N} \sum_{n=1}^{N} (c_i * x_n^{(i)})^2 = \bar{C} \tag{5.2}$$

$$\sum_{n=1}^{N} (c_i * x_n^{(i)})^2 = N\bar{C} \tag{5.3}$$

$$c_i^2 \sum_{n=1}^{N} (x_n^{(i)})^2 = N\bar{C} \tag{5.4}$$

$$c_i = \sqrt{\frac{N\bar{C}}{\sum_{n=1}^{N} (x_n^{(i)})^2}} \tag{5.5}$$

The transformed dataset exhibited the following distribution of feature values:

To compress the data into the $(0, 1)$ range necessary for classical activation functions to operate in, we then ran the c_i-normalized data through the *tanh* function. We observed that the most significant proportion of available activation after applying the normalization are concentrated in the region, as evidenced by the histogram of activations:

Note the beta distribution-like “hump”, which gives us an intuition about what range of activation means that the neuron can safely be interpreted as “switched on” when defining sparse priors.

5.1.3 ImageNet features for retrieval

Aside from using the image features as a representation of images (above which we will build other representations, to reduce dimensionality), we wish to use the features as a *retrieval search space.*

First, we need to establish that visually (or functionally) similar images are represented close to each other in this space. This property of the image feature space is not obvious. The top 1000-neuron layer was trained with one-hot vectors of image categories, so at that level, similar vectors mean the image belongs to similar semantic categories. However, at the previous level, dissimilar patterns of activation may lead to similar activity in the classification layer, and similar patterns to different classification. (Imagine all active neurons except for one have a very weak connection to all output neurons, and in a similar example this one neuron is switched off. The original network should, of course, not permit such degenerate cases of practically useless neurons; this example serves merely to illustrate that the meaning of proximity in representations may be very different.)

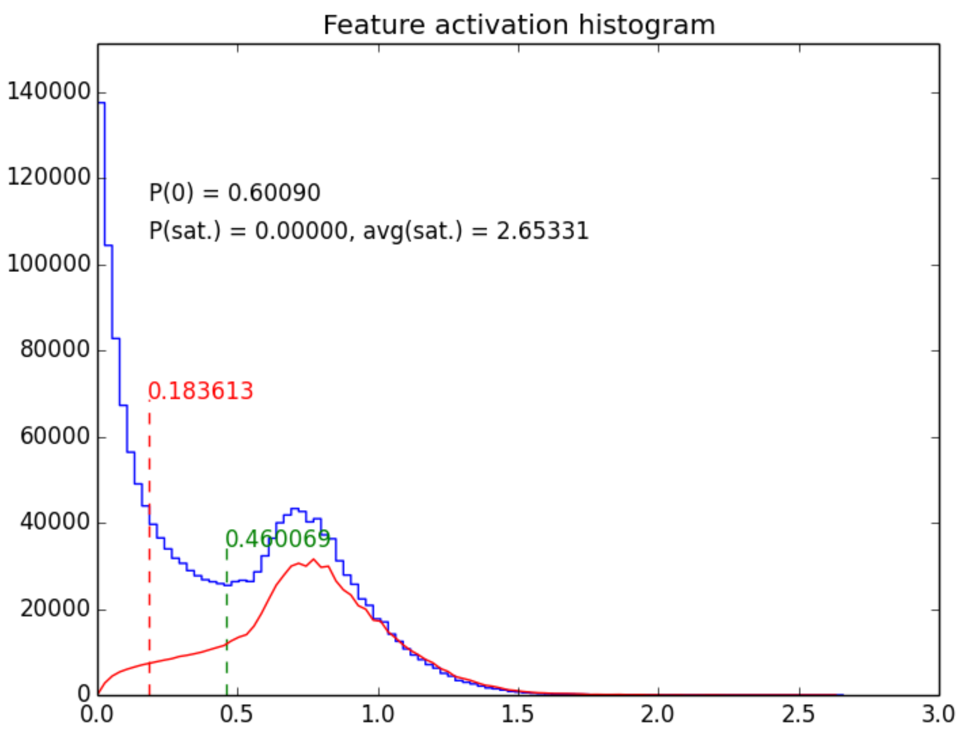


Figure 5.2: Histogram of feature values for the normalized image data. The red line represents how much "total activation" is availabe in the dataset through feature values in the given range. The

The easiest way to check that proximity in this feature space is useful is by inspecting retrieval examples:

More examples are given in section 7.2.

5.2 Text Processing

The role of text preprocessing is essentially selecting words that are relevant to the task. Preprocessing consists mainly of standart NLP techniques for battling data sparsity: lemmatization, part-of-speech filtering, removing infrequent and too frequent words. Also, we experimented with retaining as features only words that occur in the beginnings of texts, assuming that the first few paragraphs of an article sum up its message adequately, and with transforming the term-document matrix using Tf-Idf transformation.

Lemmatization and morphological tagging was done using the MorphoDiTa tagger [77], which uses the UFAL positional tagset [32]. All text experiments were done over lemmas. We experimented with retaining autosemantic words (content words: nouns, adjectives, verbs and adverbs) and subsets of those. The tagger also handled segmentation into sentences.

It was found that omitting adverbs was also beneficial, as the majority of frequent adverbs was also comprised of low-content words such as "how", "a lot", "yesterday", etc.

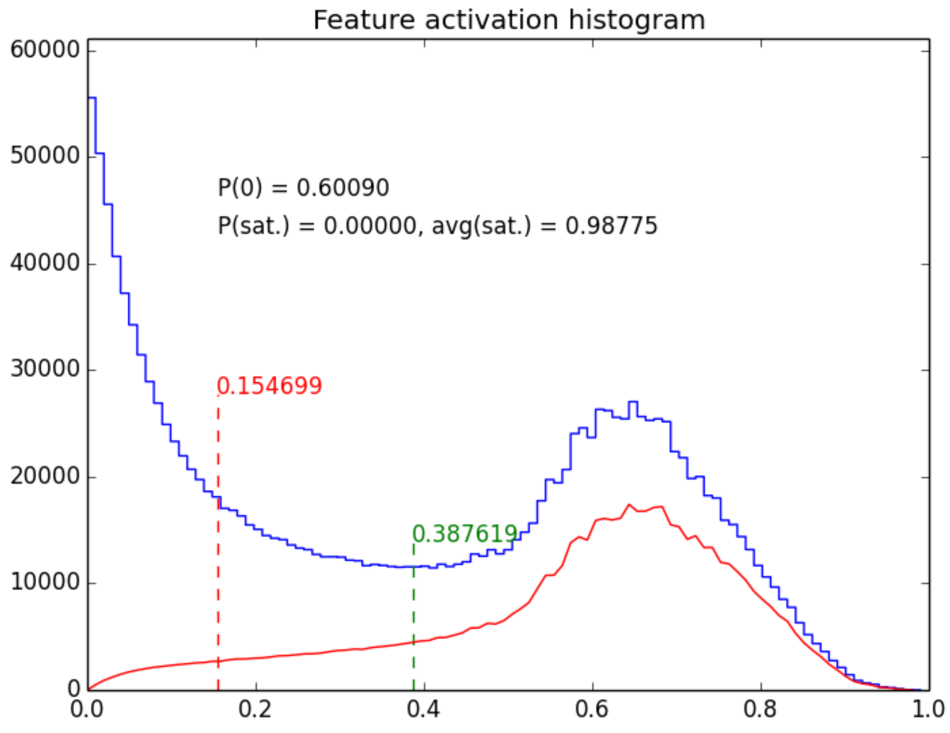


Figure 5.3: Histogram of feature values after applying the sigmoid.

Next, because journalistic texts most often condense the gist of their message in the first few paragraphs, we experimented with retaining only a certain portion of the text from the start. We tried both a fixed amount of sentences (5, 10, 20) and a proportion of the entire text (0.2, 0.3). However, to differentiate signal from noise, we compute for the retained words frequencies *from the entire document* - this way, words that are pertinent to the document's actual topic gain comparatively greater influence than words that occur more or less randomly, like figures of speech journalists sometimes like to use in the opening paragraph.

At this stage, we apply the standard Tf-Idf transformation across the corpus, and normalized each document vector to unit Euclidean norm.

Finally, we applied frequency filtering, discarding k' most frequent words (general terms and domain-specific irrelevant terms: ”to be”, ”to have”, ”time”, ”year”, ”author”) and retaining only the next $K - k'$ most frequent words. The value of k' was set to 10, since in positions around 11 - 20, words like ”man” or ”child” often appeared, which are already relevant to selecting images.

The extracted text features and their values with 10 sample sentences:

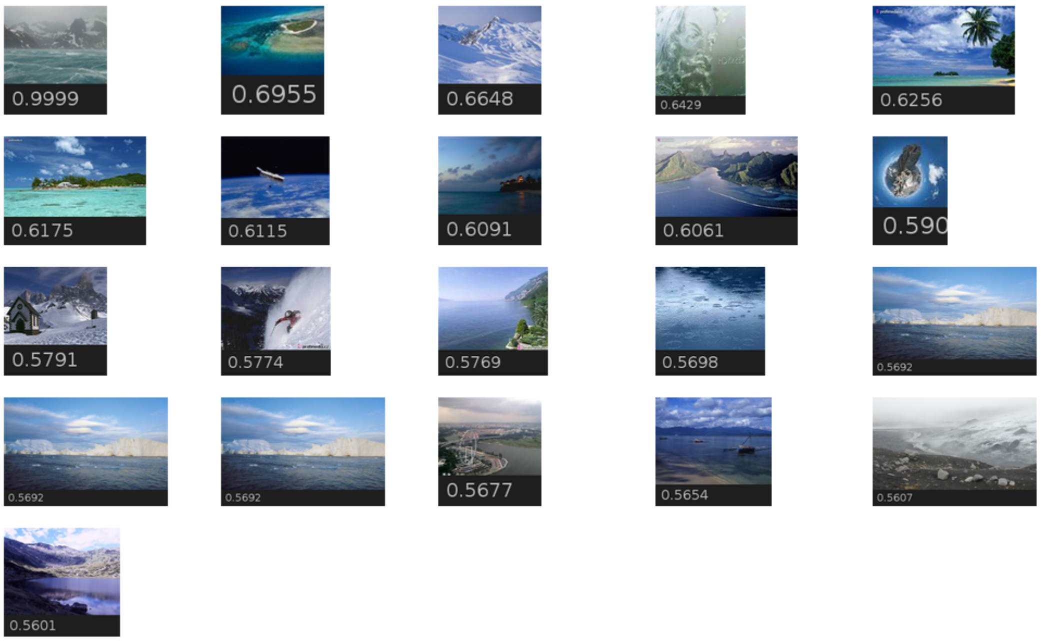


Figure 5.4: Similarity search in UCov. feature space. (Top left image is query.)

guitar	0.463
musical	0.386
learn	0.232
difficult	0.215
firm	0.154
try	0.154
willpower	0.154
everyone	0.108
want	0.081
person	0.040

Is it difficult to play the guitar?
Most people are asking me whether it is difficult to play the guitar. I say it's not. Everyone has something inside they can utilize for themselves. Learning to play well takes strong will. I found out that I had musical talent. I tried to play the guitar so much and after a long effort, it was worth it. (. . .)

Table 5.1: An example of 10 most prominent extracted text features

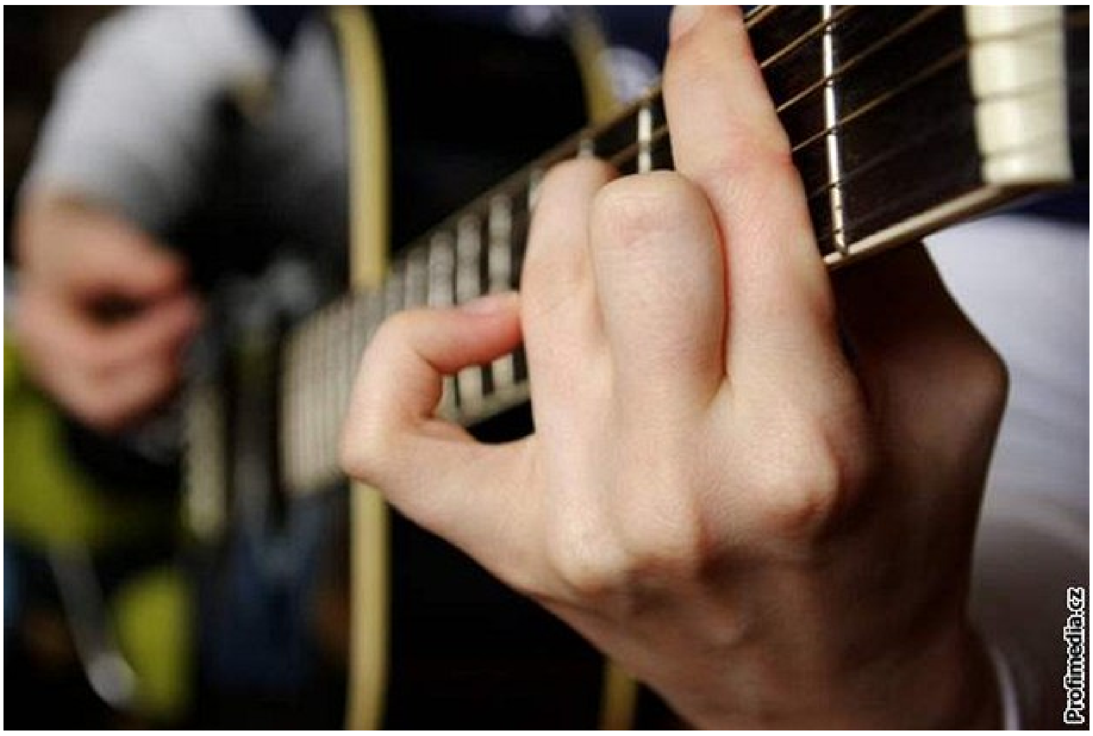


Figure 5.5: Image originally accompanying the article

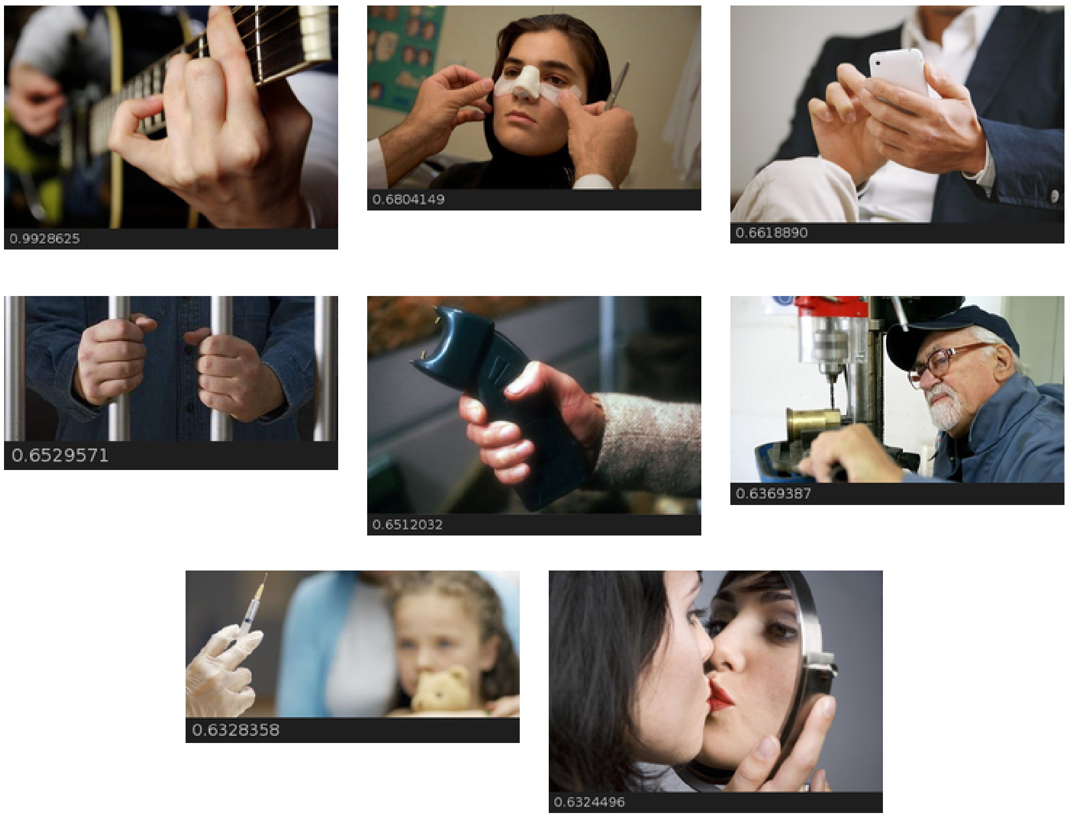


Figure 5.6: Unfortunately but not too surprisingly, in this case, the image features seem to have focused on the more prominent hand.

6. Deep Learning experiments

In this chapter, we describe the multimodal model(s) we used and the experimental setup[1] and the training procedure.

6.1 Multimodal model

Following the architecture of Srivastava and Salakhutdinov [73], we train two stacks of unsupervised models and a joint "roof" layer. The stacks are trained using the greedy unsupervised layer-wise pre-training procedure [33].

The entire learning process is unsupervised: the joint layer is also learned as a generative (or pseudo-generative, for autoencoders) model with the inputs set to a concatenation of the top-level image-only and text-only representations of the training text-image pair.

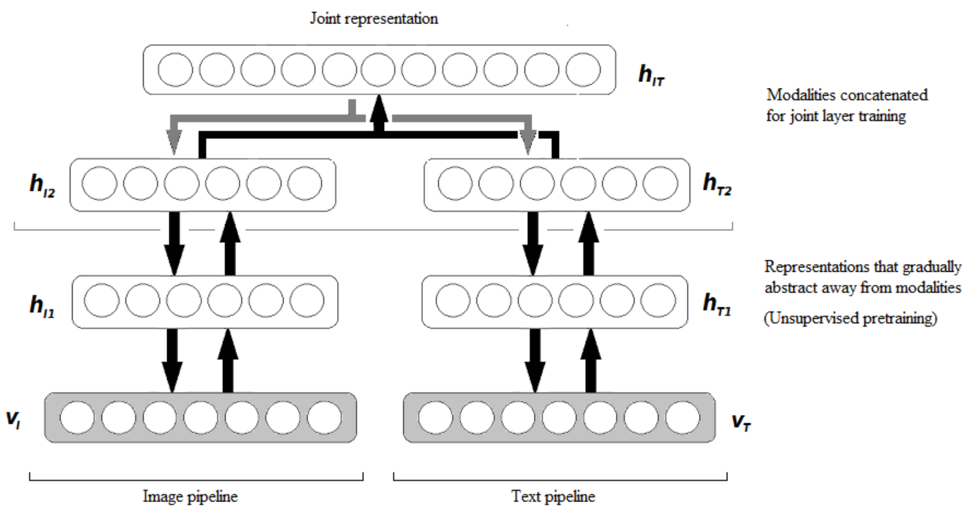

Joint representation
h_{IT}
Modalities concatenated for joint layer training
h_{I2}
h_{T2}
h_{I1}
h_{T1}
Representations that gradually abstract away from modalities
(Unsupervised pretraining)
v_I
v_T
Image pipeline
Text pipeline

Figure 6.1: The architecture of the multimodal model. (The optimal number of the hidden layers is not necessarily 2.)

To obtain an image, the text is first transformed to successive representations in the text processing pipeline. The text inputs to the joint layer are then fixed at the activations obtained from the text pipeline and the values of the joint layer and other image features are sampled from the model distribution conditioned on the clamped text representation. (Because the lower-level text representations are conditionally independent on the joint layer and image features given the topmost text layer, we can disregard them at this stage.)

Once a sample of the image features is obtained, it is used as a similarity query in the vector space of all available images in the data. We can choose the

[1]Because we were unable to learn a good representation for the text data, the full multimodal model was in practice only run once, to verify the expected result: retrieving the same – or very nearly same – images for each document. See 7.

level of representation at which we want to search: after training the image stack, we can transform the original image data and build a similarity index using the learned representations.

To obtain samples of top-level image activations, we should sample from the full model, as described in [73]. We approximate this procedure by only sampling the top layer, forgetting about the rest of the stack. This approximation allows us to easily plug models *without* a straightforward probabilistic interpretation into the image pipeline (as well as the text pipeline). To obtain the "proposed image", we run an alternating Gibbs chain for K steps between h_{IT} and h_{I2}, h_{T2}, with the clamping expressed as setting the proposal distribution $p^{(k+1)}(h_{T2} \mid h_{IT}^{(k)})$ to $\delta(h_{T2}^{(0)})$,[2] the distributions $p^{(k)}(h_{IT} \mid h_{I2}^{k}, h_{T2}^{k})$ and $p^{(k+1)}(h_{I2} \mid h_{IJ}^{k})$ stay unchanged. We initialize the chain at the "mean representation" $h_{I2} = \frac{1}{\mathcal{D}} \sum_{j=1}^{|\mathcal{D}|} h_{I2}(j)$.

(Unfortunately, because of the failure of training a text representation, we were not able to test whether the sampling procedure approximates sampling from the full conditional $p(h_{I2} \mid h_{IT}, h_{I1}, v_I)$ well enough.)

6.2 Image models

Over the image modality, since the image features are already the results of an advanced neural network [45], the task is less a question of finding a meaningful representation and more of a straightforward dimensionality reduction task.

We experimented both with Autoencoders and Restricted Boltzmann Machines. The standard sigmoid activation $\sigma(x) = \frac{1}{1+e^{-x}}$ was used. The image stack was trained using Denoising Autoencoders with zero-masking noise and the corruption level set to 0.3. The first layer was set to 1000 neurons, the second to 250. We used cross-entropy (defined in eq. 2.4) as the reconstruction function.

We applied both denoising autoencoders and RBMs.

6.3 Text models

We tried:

- Restricted Boltzmann Machines
- Sparse RBMs
- Replicated Softmax
- Denoising Autoencoders
- Sparse DAs

We tested hidden layer sizes of 2000, 1000, 250 and 100, with various activation functions: sigmoid, piece-wise approximation of sigmoid, softplus, capped ReLU and hyperbolic tangent. We also experimented with regularization: sparsity, weight decay and bias decay.

Weights were initialized uniformly from the interval $\pm\sqrt{\frac{6}{\|V\|*\|H\|}}$ and biases were initialized to 0.

[2]The δ here stands for the Dirac delta function centered at the input text representation.

6.4 Training procedure

The learning algorithm was simple minibatch stochastic gradient descent. It is more efficient computationally to use larger minibatches rather than multiple smlaler ones, thanks to thoroughly optimized matrix-matrix multiplication routines; however, smaller batches allow for faster parameter updates, especially in the beginning of training. We found a good convergence speed vs. training time per item tradeoff at batch sizes around 100. The learning rate was kept at 0.13.

The Restricted Boltzmann Machines were trained using CD-1 to estimate the gradient. The hidden layer was sampled and the mean (i.e. activation) was used for the negative particle.[3] The autoencoders were trained using gradient from backpropagation[4] computed using symbolic differentiation in theano. 8.2

All experiments were run on a Lenovo W520 workstation laptop, using the NVidia Quadro 2000M GPU with 192 CUDA cores. Mathematical operations were run using Intel's MKL library [19] optimized for the Canopy distribution [1] of the numpy [81] Python package. One epoch of training on the web-pic training dataset took between 6 and 60 seconds, depending on the complexity of the model. Without the GPU, training took about three times as long.

[3]As recommended in [?].

[4]Autoencoders are feedforward models, altough they are unsupervised, so backpropagation applies. In a sense, they are "self-supervised" by the input data.

7. Results

We would very much like to report meaningful MAP (mean average precision) results. However, all text representations we trained found a degenerate solution, representing all documents with the same (or extremely similar/correlated) vectors. Therefore, as the image features are conditionally independent on the text inputs given the text representations, the *same images will be retrieved for all documents.* We have trained and run one full multimodal model and it confirmed that this was indeed the case. [1] The MAP@10 (10 pictures is a reasonable number that a journalist can process as a query result) was 0.00.

This problem was caused squarely by the text pipeline: the neurons of the topmost text layer are clamped when sampling the joint layer, so they will always direct the top-level Gibbs chain in the same direction, regardless of what text was given. On the image data separately, we were able to train denoising autoencoders with retrieval properties comparable to retrieval in the original 4096-dimensional space. We therefore focused our efforts on training usable text representations.

We were unfortunately unable to find settings or models that would overcome these problems. In the rest of this chapter, we will present an analysis of the networks' behavior.

7.1 Analyzing text processing failures

We do not claim to have complete understanding of why learning document representations failed. However, we will at least attempt to analyze the behavior of the model and try to find some causes of the failure, and suggest remedies.

We will refer to the activation of the i-th input neuron in data item n as $x_i^{(n)}$.

The following preprocessing steps have been applied:

- Part of Speech filtering, retaining only nouns, adjectives and verbs
- Positional filtering: only words that appeared in the first 20 % of sentences from each article were used, but their frequencies were counted over the whole document.
- Capitalized lemmas were removed. This was especially useful for removing pervasive geographical ("Czech Republic") and reporter names.
- Tf-Idf processing was applied and the resulting *data* vectors (rows in the visualizations below) were scaled to unit norm.

7.1.1 Data visualization

We will first visualize the text data. We plot the dataset heatmap and the histogram of activations. The "starry sky" heatmap shows us what the samples generated from a well-trained representation should look like (Fig. 7.1).

When looking at the activations in detail, we see that there is a bit less black than it would seem at first sight (Fig. 7.2).

[1]The model used one hidden layer for each modality and one joint layer. The hidden layer for images was a Denoisign Autoencoder with 2000 hidden neurons. For the text and joint layers, we used a Restricted Boltzmann Machine. Both were trained for about 120 epochs, until – apparently only temporary – convergence.

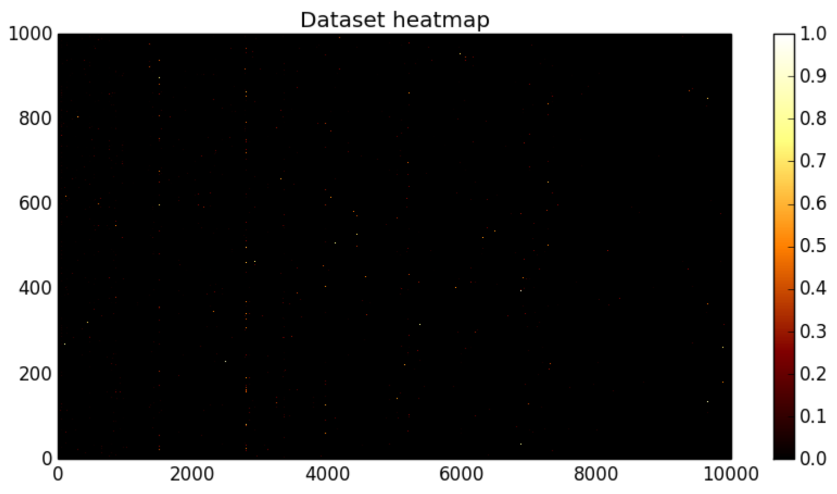


Figure 7.1: Plotting the activation of the dataset

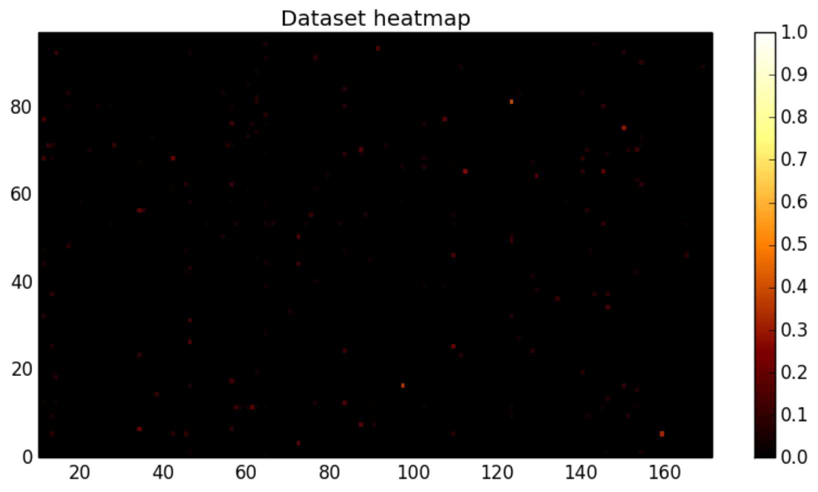


Figure 7.2: A more detailed view of the activation on a small part of the dataset

There are, of course, naturally occurring correlations between features. The clearly noticeable columns (Fig. 7.3) correspond to the words "scientist" and "research".

The (nonzero) feature activations over the entire dataset approximately follow the distribution visualized in (Fig. 7.4).

7.1.2 Training a model

We will run SGD for a 250-neuron Restricted Boltzmann Machine on this dataset to illustrate how the network's distribution $P(x, v)$ evolves.

The data on which we will be tracking the model's behavior is visualized in (Fig. 7.5). This is a part of the heldout data which do not contribute to gradient descent.

The next figure 7.6 is a visualization of the corresponding hidden unit activation at initialization. Since $\sum_{i \in I, j \in J} \mathbf{W_{i,j}}$ is close to 0 for any index sets I, J due

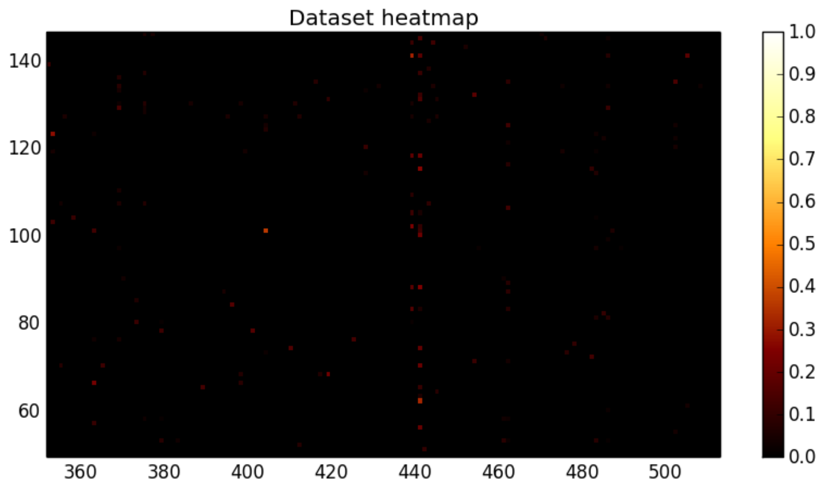


Figure 7.3: Correlated features

to the weight initialization procedure (see 6.3), the activation falls very close to 0.5.

The reconstruction at initialization is also very close to the center of the sigmoid, as illustrated in (Fig. 7.7).

After one pass over the training data, the CD-1 negative particles look like (Fig. 7.8).

After seventy nine more passes, the negative particles look quite similar, as evidenced by (Fig. 7.9)

Either the gradient has very quickly reached a plateau and cannot move away, or there is a degenerate local minimum.

We first worked under the assumption that generating negative particles from the unigram distribution $P(v \mid \mathcal{D})$ is some sort of a local minimum that is hard to leave because the counter-examples – where less common words play a major role and frequent words don't – are much less frequent, thus relegated to the level of background noise, and the model will much rather optimize itself for the frequent words.

We tried various regularization to force the model to change the kind of representaiton it prefers: notably, the generalized notion of *sparsity*. By setting a target mean activation for all features alike, we hoped to break up the "columns" that stretched throught the data representation. Adding sparsity to a model is easy: the loss function simply acquires a sparsity term for each neuron.

$$S_\rho(x) = \rho \; log \; \frac{\rho}{\hat{\rho}} + (1 - \rho) \; log \; \frac{1 - \rho}{1 - \hat{\rho}} \tag{7.1}$$

This is the KL-divergence of a Bernoulli distribution with mean ρ to a Bernoulli distribution with the empirical mean $\hat{\rho}$.

However, adding sparsity also did not produce a desirable representation. The activity of the network hints more at modeling document, instead of feature, magnitudes.

Based on the observation in 7.2.2, *we are currently leaning more towards the variant that this area of modeling is a large plateau of the loss function that has*

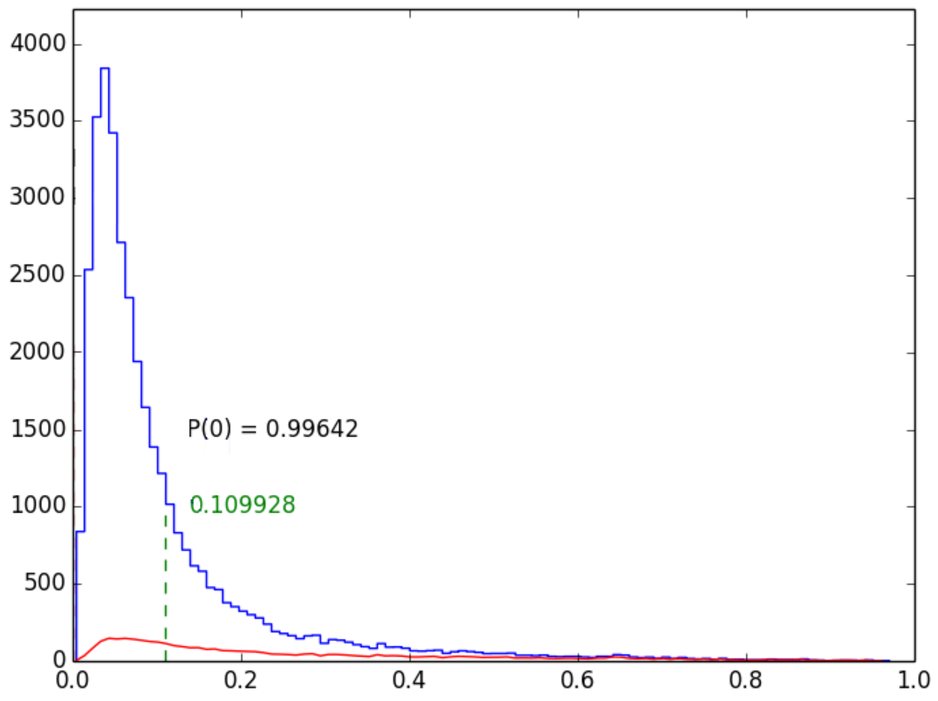


Figure 7.4: Distribution of feature weights. The red line corresponds to how much of the total activation available is concentrated in such (n, i) that their activations $x_i^{(n)}$ fall within the given range on the X axis. (If the histogram was uniform, the red line would be following $Y = X$.) The term $P(0)$ refers to the sparsity: the proportion of (n, i) s.t. $x_i^n = 0$.

Figure 7.5: The heldout data on which we inspect model behavior.

to be crossed. Instead of a problematic loss function, we have a slow learning algorithm.

Again, the culprit is most probably the sparsity of the trainig set. The interaction is simpler than competition: infrequent features probably just learn *that* much slower. The model first learns to generate frequent words from the empirical categorical distribution very fast, because it has enough training examples to "discover" the subspace of projecting to only these less sparse features and tries to move there. It takes many more iterations for a word that is seen just eight or ten times in the dataset to exert at least a slight "pull" on the model. Learning in the subspace of frequent features simply goes a lot faster than for the infrequent examples.

The appropriate remedy here would be a more sophisticated, faster learning algorithm. If we are dealing with a smooth plateau, Conjugate Gradient may help span larger distances, as it determines the optimal learning step using line search. If the gradient is more or less consistently going in one direction (although possibly zigzagging), the rprop algorithms may help speed it up considerably, independent on the magnitude of the gradient.

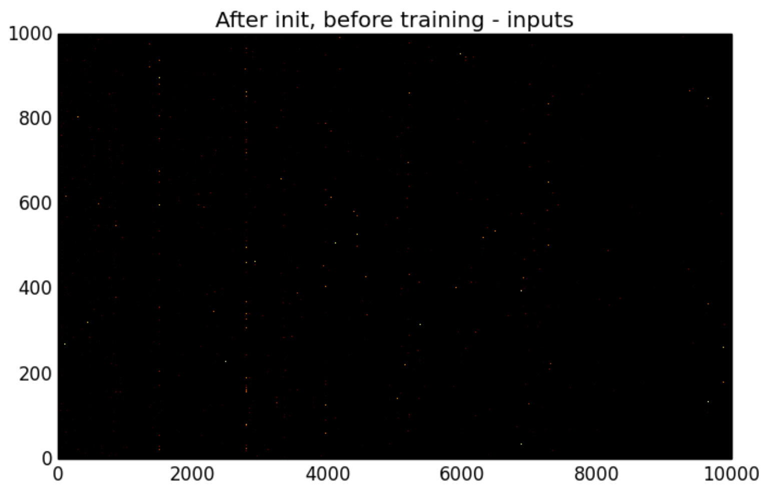


Figure 7.6: Transformation with the initial parameter values.

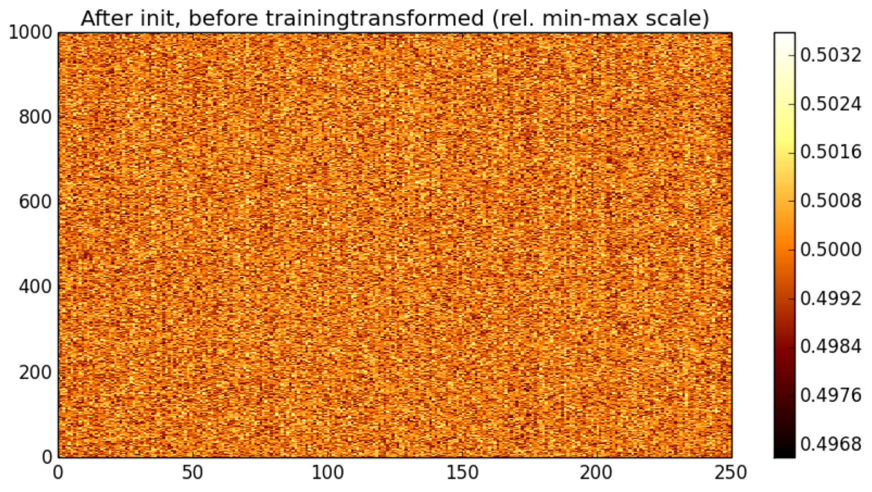


Figure 7.7: Reconstruction with initial parameter values.

(It is also possible that a part of the problem lies in the Contrastive Divergence algorithm. Some problematic behaviors of CD have been outlined by Breuleux et al. in [13].)

7.2 Image results

Training the image stack was a matter of dimension reduction, as a state-of-the-art distributed representation learned by the ImageNet CNN [45] was already available. For this task, autoencoders are a natural choice; we also tried RBMs to see how they would compare.

Since we cannot evaluate the image pipeline in the context of a multimodal model, we at least provide a qualitative evaluation: comparing the output of a similarity query at various levels of representation and try to find an interpretation of similarity in the image representation spaces. Since visualizing the action of

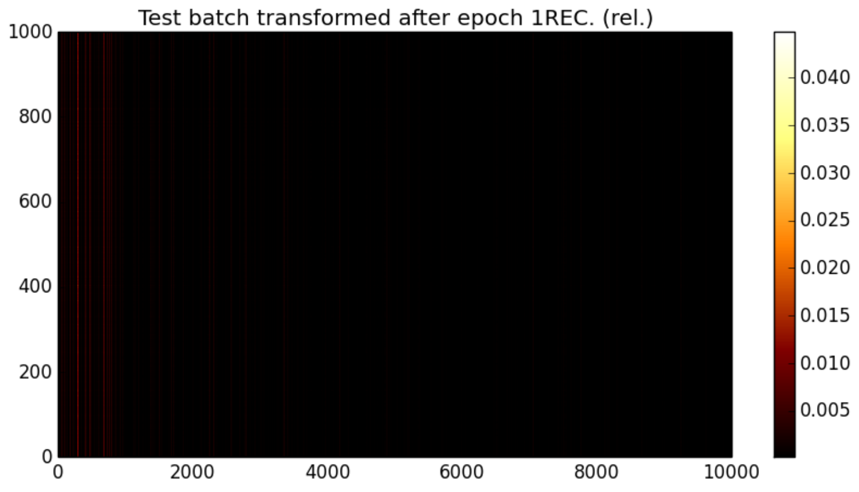


Figure 7.8: Negative particles after the first epoch.

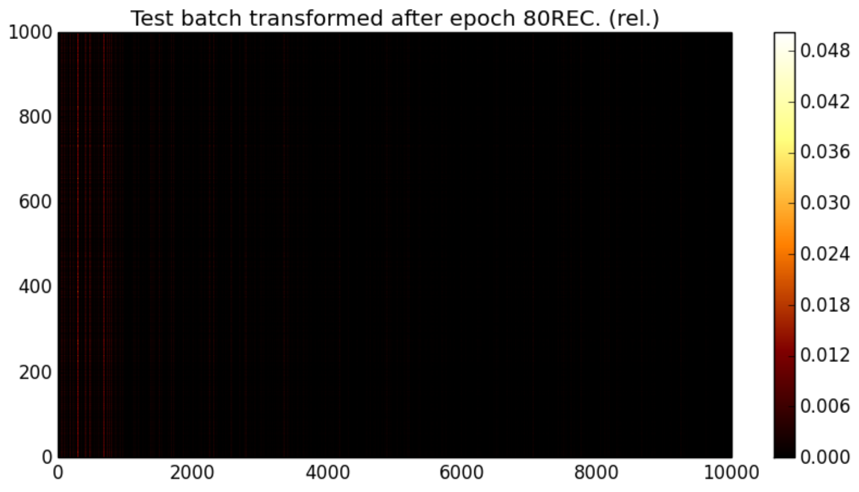


Figure 7.9: Negative particles after the 80th epoch.

neural network is difficult and an open research problem [25], we are going to have to “resort” to our human understanding of the examples.

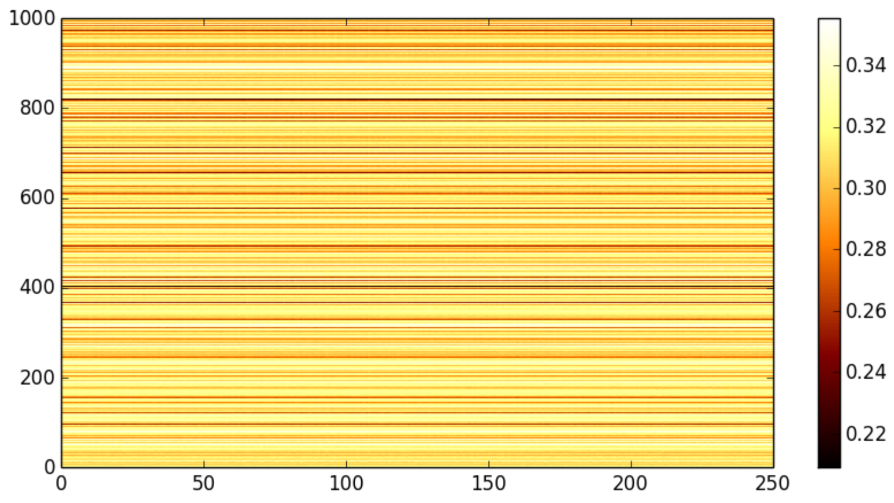


Figure 7.10: Representations learned after 40 epochs with a sparsity target set to 0.3. Notice the range of heatmap colors. Setting different sparsity targets led to the same structure learned, only at a different scale.

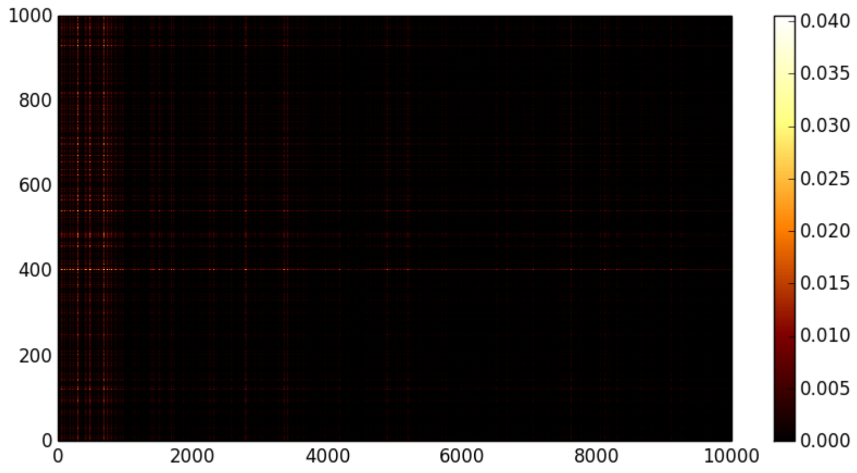


Figure 7.11: Negative particles from the trained model. Notice the grid-like pattern: it probably represents some saddle-point in a transition from horizontally-degenerate to vertically-degenerate states.

7.2.1 Inspecting retrieval

We report the original image and 9 most similar in the given representation. Similarity is measured using the cosine distance. The query image is always the top left thumbnail.

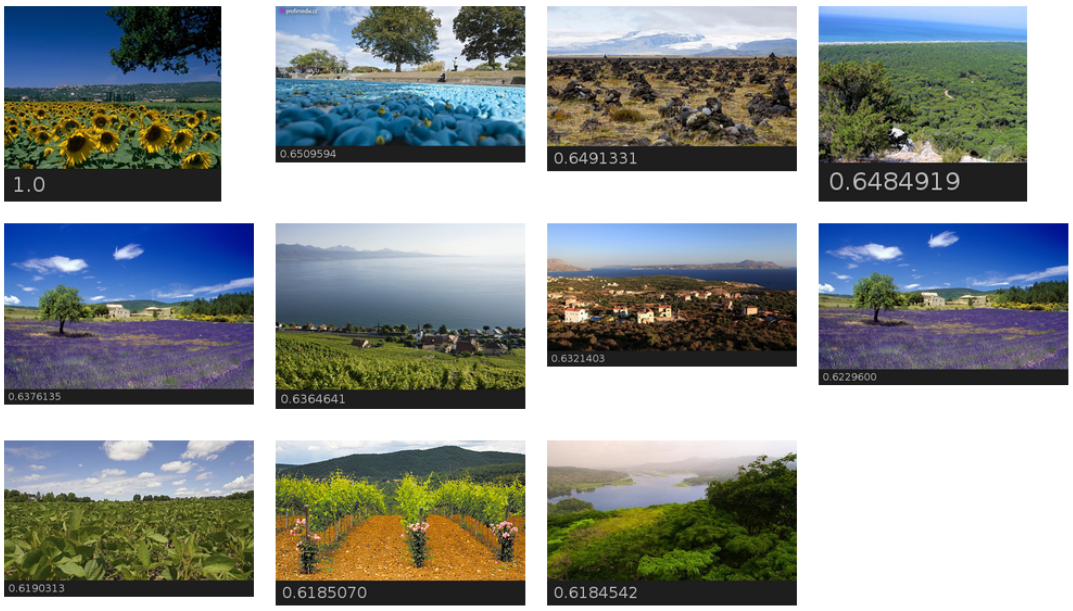


Figure 7.12: Original 4096 features + c_i scaling + *tanh* sigmoid

First, we compare a scene of several well-defined textures. It seems that small differences in texture have a much greater influence than a change of color: the 1000-neuron RBM (Fig. 7.15) retrieves a green field in between two practically identical yellow fields. Also, the image of dark patches in ochre grass appears very close to the top in 4 of 5 of the models, including the original, mimicking the sunflowers; this would suggest that the image features encode also the presence of "macro-textures", regular edge structures on a larger scale.

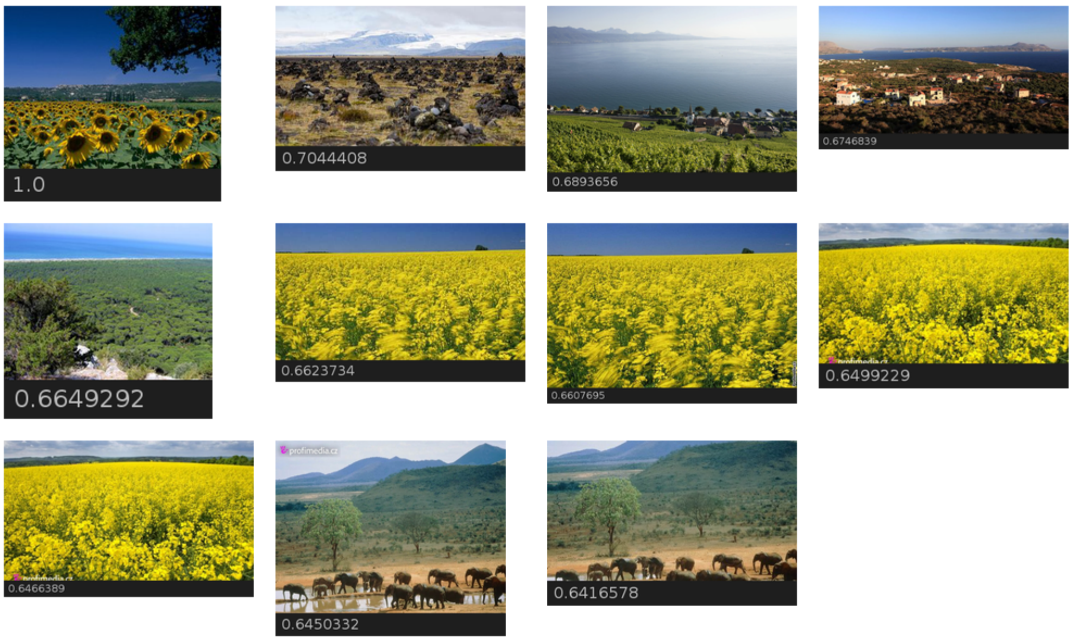


Figure 7.13: Denoising autoencoder, 1000 neurons

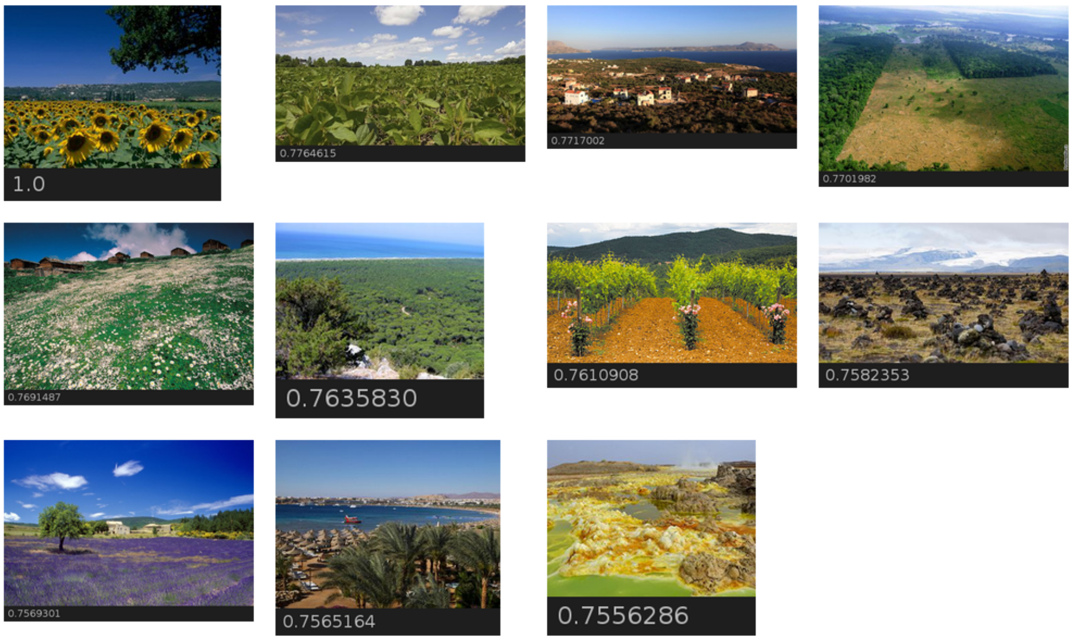


Figure 7.14: Stacked Denoising Autoencoder, 1000 + 250 neurons

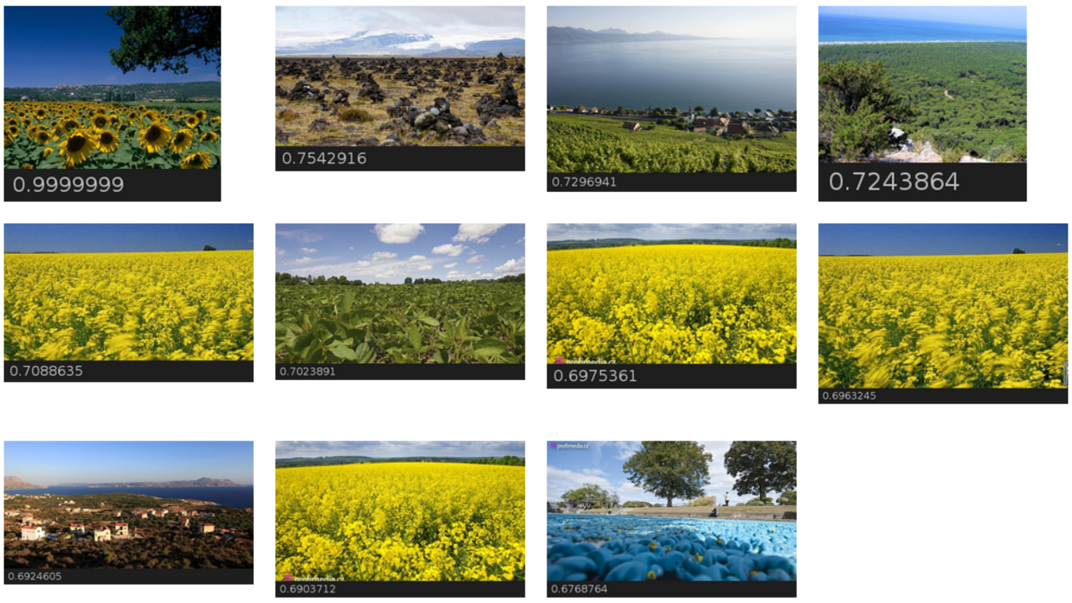


Figure 7.15: Restricted Boltzmann Machine, 1000 neurons

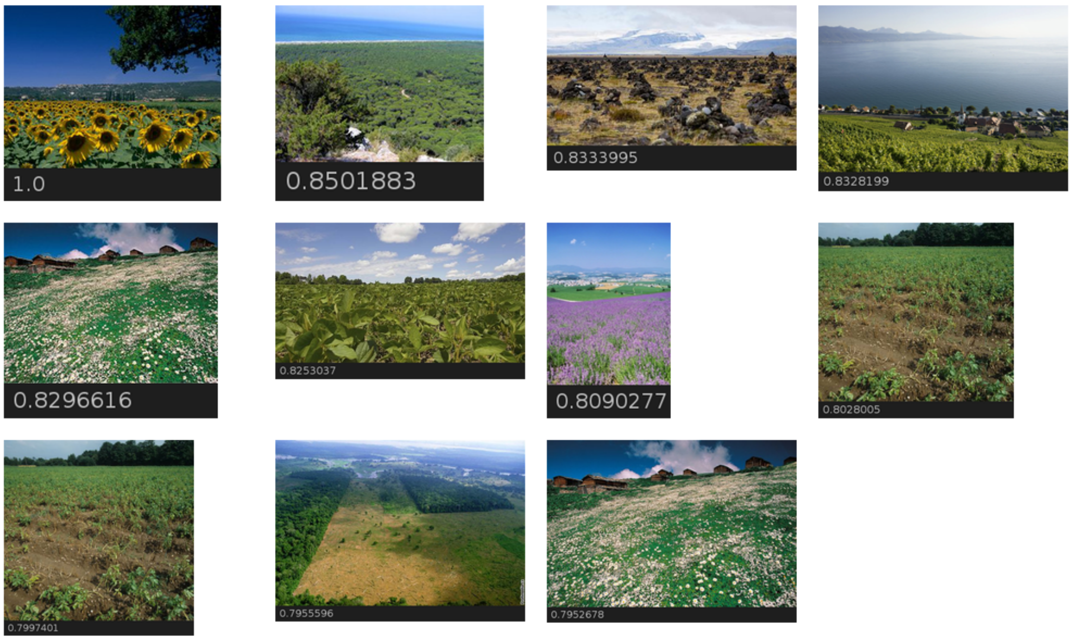


Figure 7.16: Stacked Restricted Boltzmann Machines, 1000 + 250 neurons

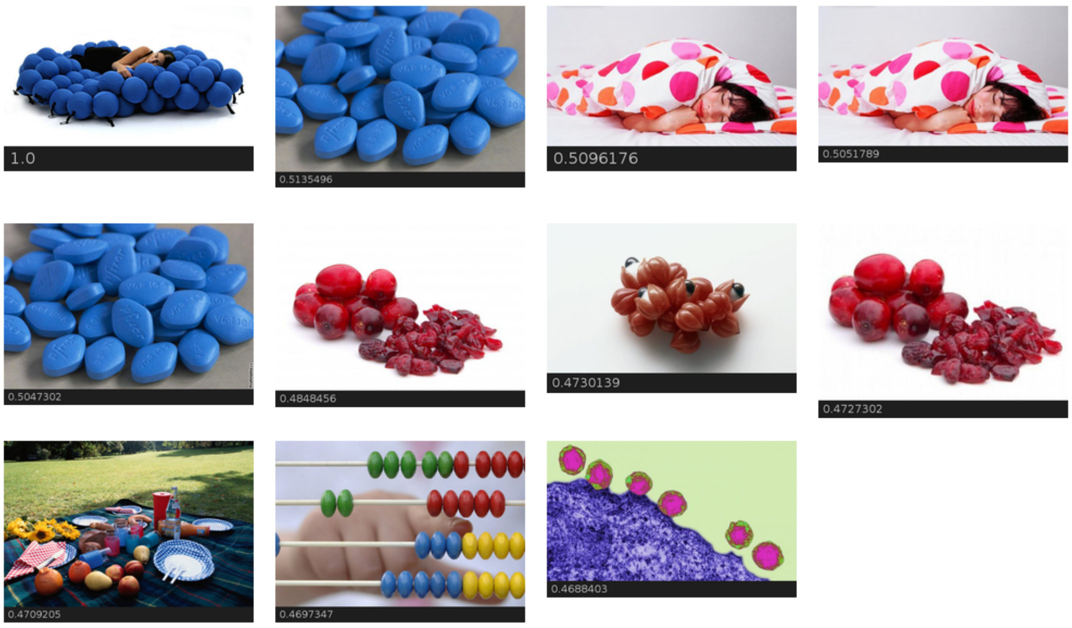


Figure 7.17: Original 4096 features + c_i scaling + *tanh* sigmoid

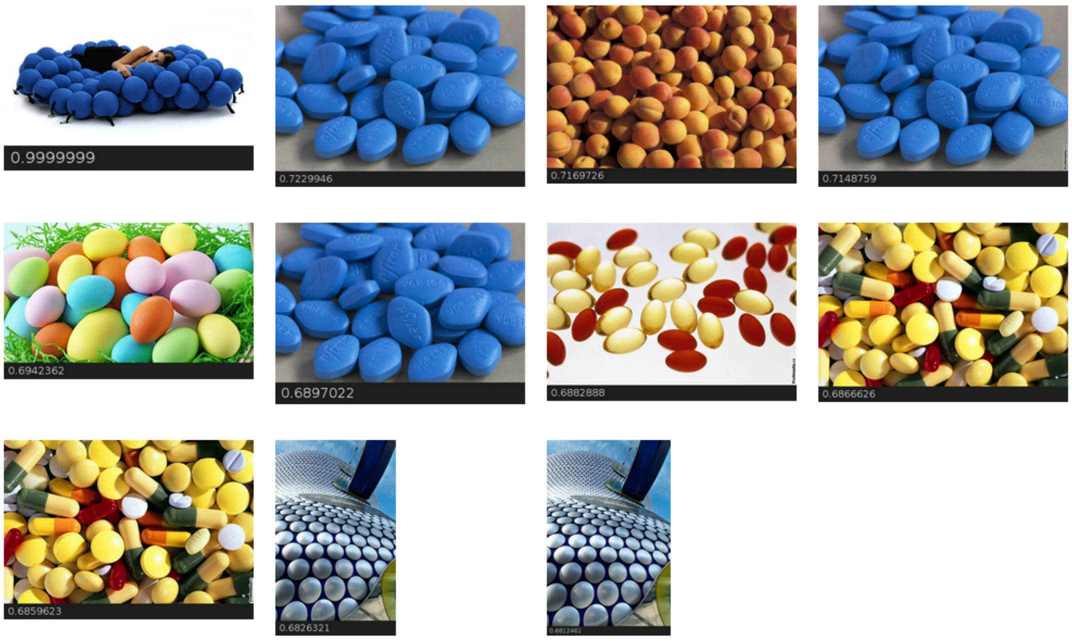


Figure 7.18: Denoising autoencoder, 1000 neurons

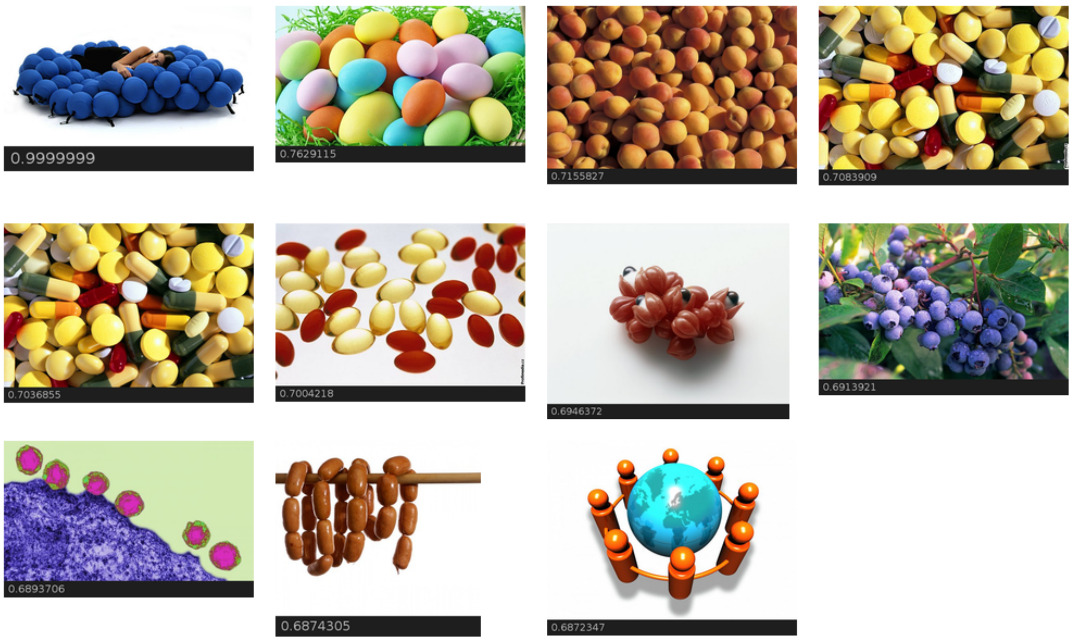


Figure 7.19: Stacked Denoising Autoencoder, 1000 + 250 neurons

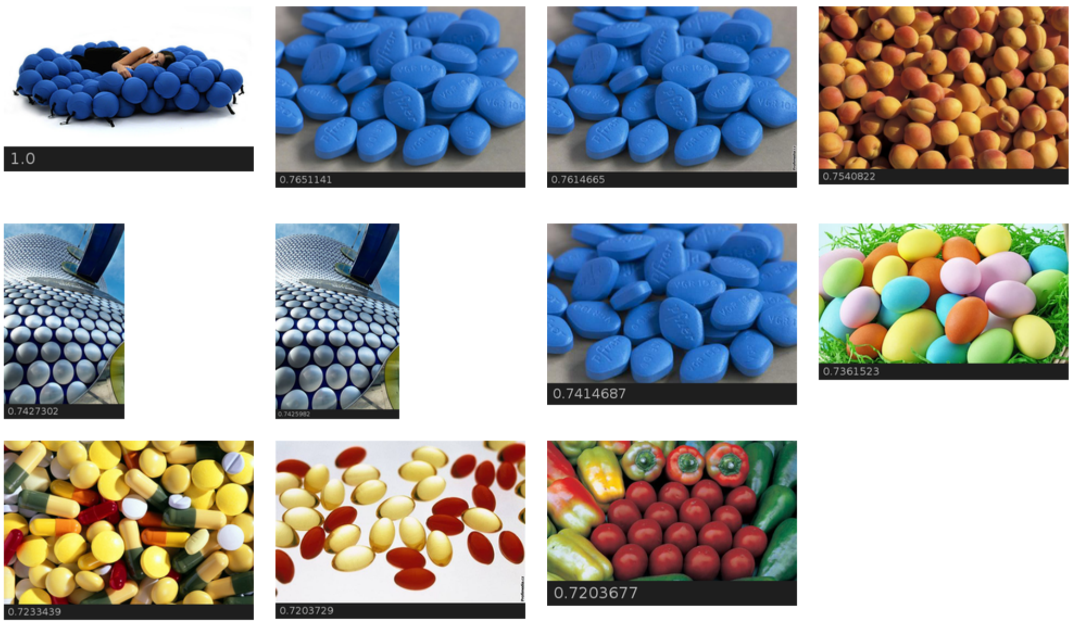


Figure 7.20: Restricted Boltzmann Machine, 1000 neurons

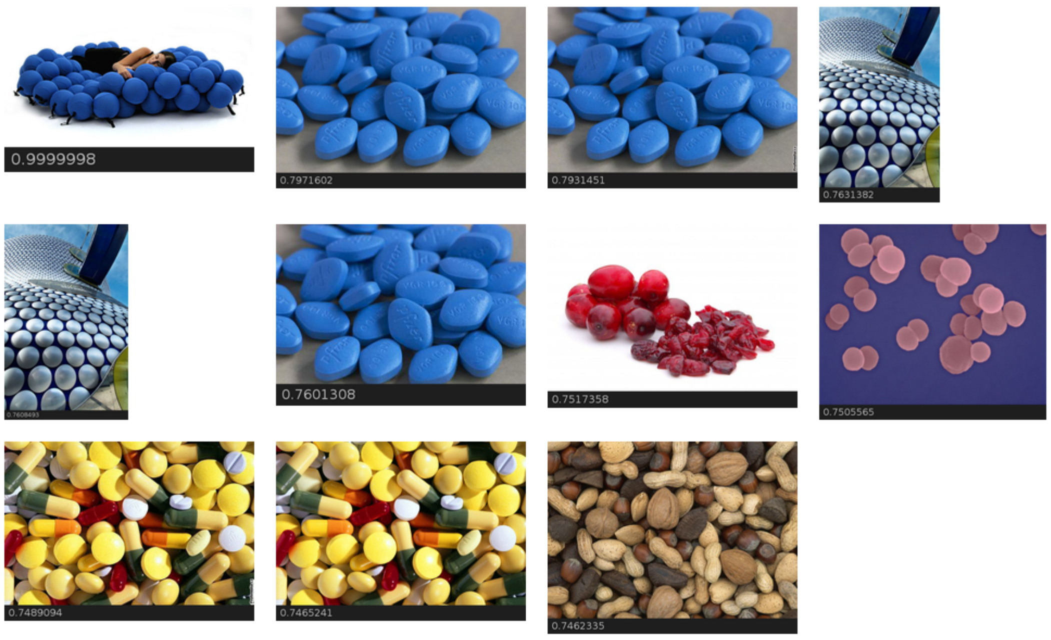


Figure 7.21: Stacked Restricted Boltzmann Machines, 1000 + 250 neurons

The second example shows various types of similarity the models are able to notice practically orthogonal to each other: color, texture, positioning, scaling. Texture – as was seen in the first example – seems to be dominant. However, it is also an example where information is lost in transformation to the higher-level representations and as a consequence, a potentially appropriate picture is missed. The upper-right picture of a woman sleeping (Fig. 7.17) is unfortunately lost in all the learned representations.

In the third set of examples, we can see how the lower-dimensional representations may help discover interesting new candidates. Both the stacked RBMs (Fig. 7.26) and autoencoders (Fig. 7.24) have discovered the image of the man with the cart and the autoencoders have also found a new image from the inside of a cave. While one example is definitely not proof, it shows that one can *gain* retrieval performance from training further layers above the high-quality image features from Krizhevsky et al. [45], depending on how the model is trained.

The fourth and last set of examples of retrieved images uses a query image with more clutter. It seems that the straight lines of the tripods in the query image are a dominant feature for the original 4096-neuron representation (Fig. 7.27), together with the presence of people, but not as much in the other models: the 3rd, 4th and 7th pictures of the top ten most similar images in the original image feature space are not found among the top 10 anywhere else. On the other hand, the 4096-1000-250 autoencoder (Fig. 7.29) model discovers a recording session and a concert that the original model did not.

(The stacked RBMs (Fig. 7.31) may start relying more on the large-scale distribution of brightness or color – the picture of soldiers looks very similar to a vertical flip of the query image if one lets the picture blur; in the same way, the Indian schoolroom is similar in the distribution of faces to a horizontal flip of the query image, and the picture of Columbus also starts making a little more sense when flipped horizontally.)

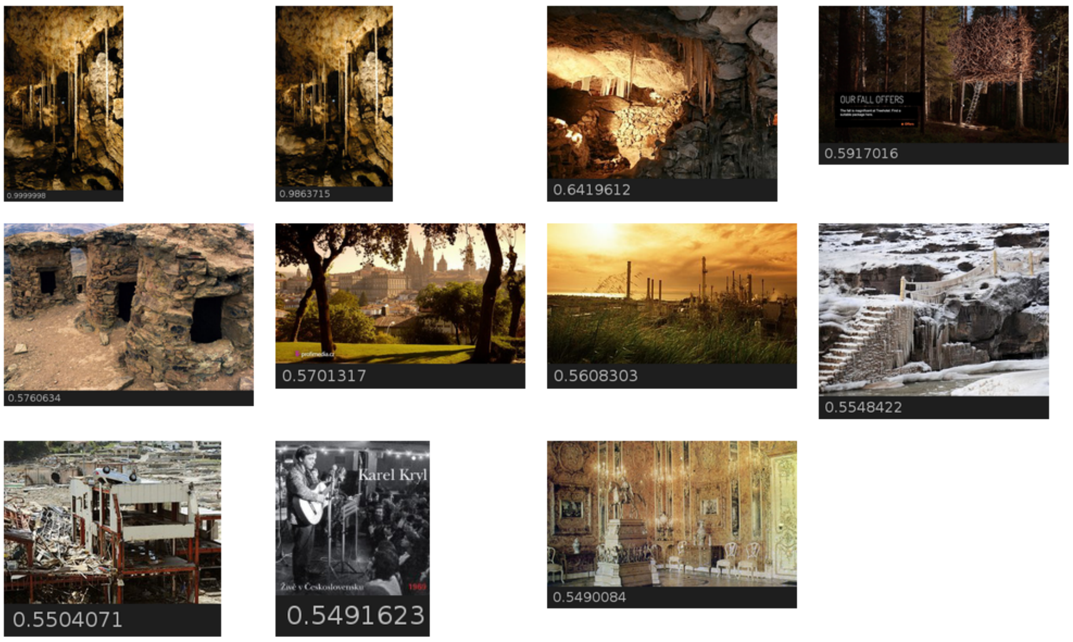


Figure 7.22: Original 4096 features + c_i scaling + $tanh$ sigmoid

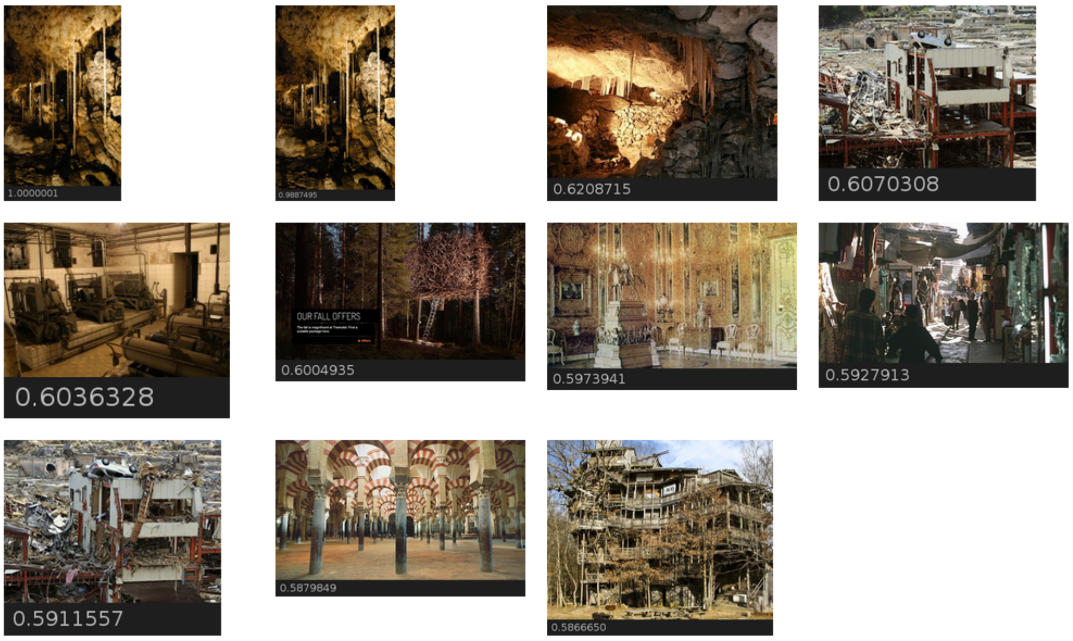


Figure 7.23: Denoising autoencoder, 1000 neurons

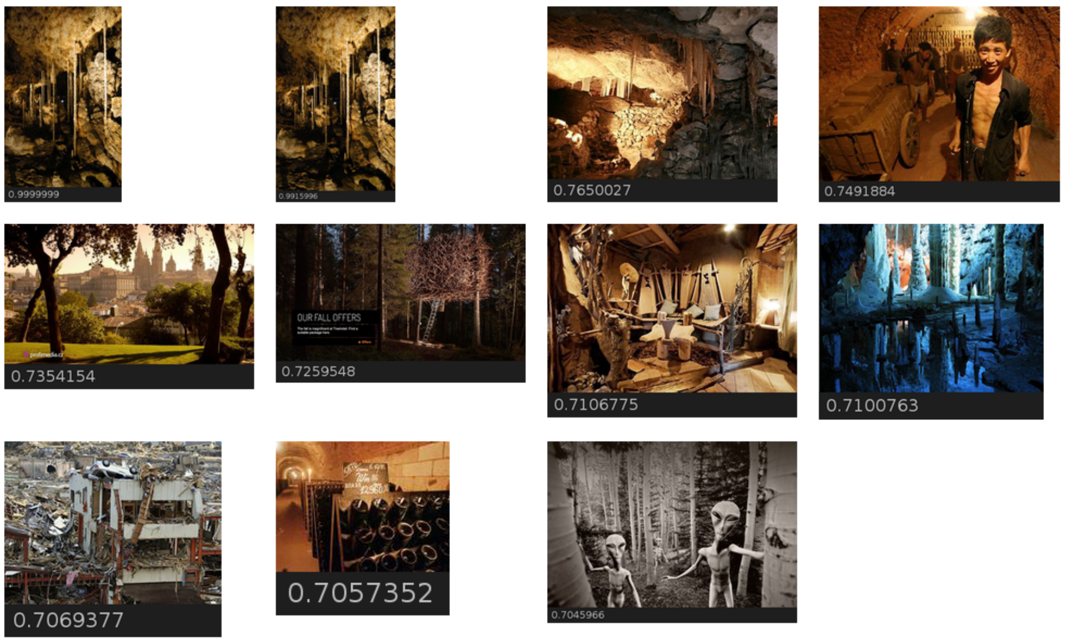


Figure 7.24: Stacked Denoising Autoencoder, 1000 + 250 neurons

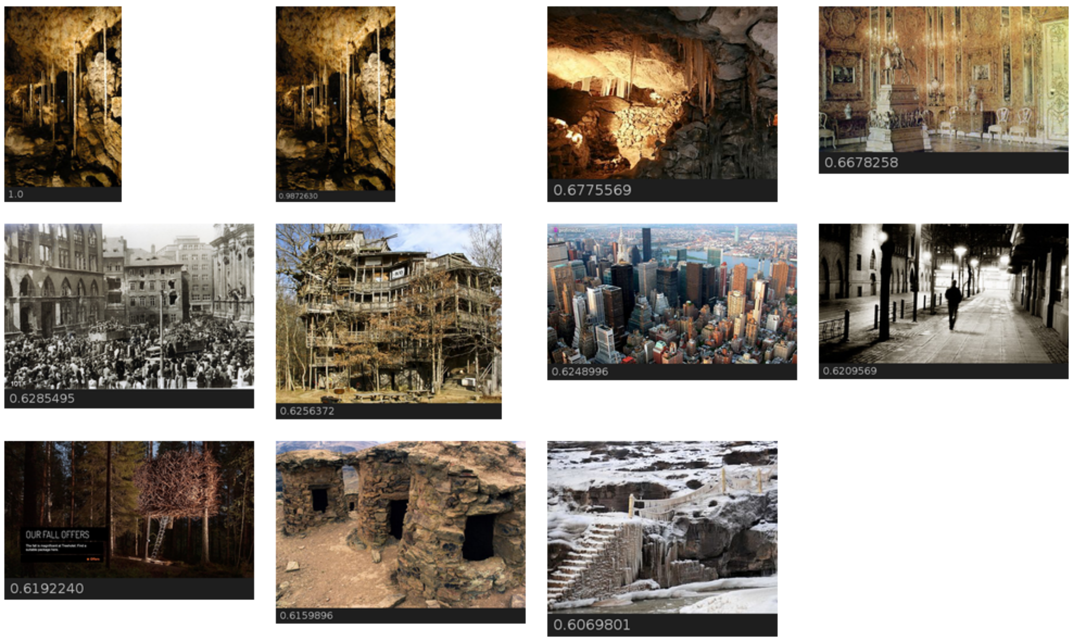


Figure 7.25: Restricted Boltzmann Machine, 1000 neurons

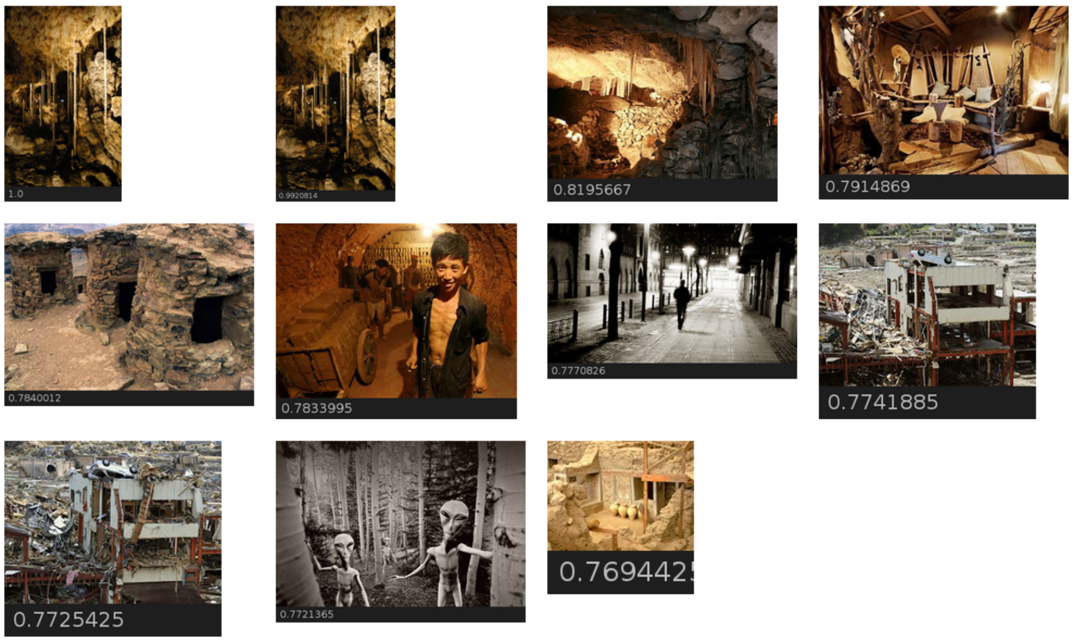


Figure 7.26: Stacked Restricted Boltzmann Machines, 1000 + 250 neurons

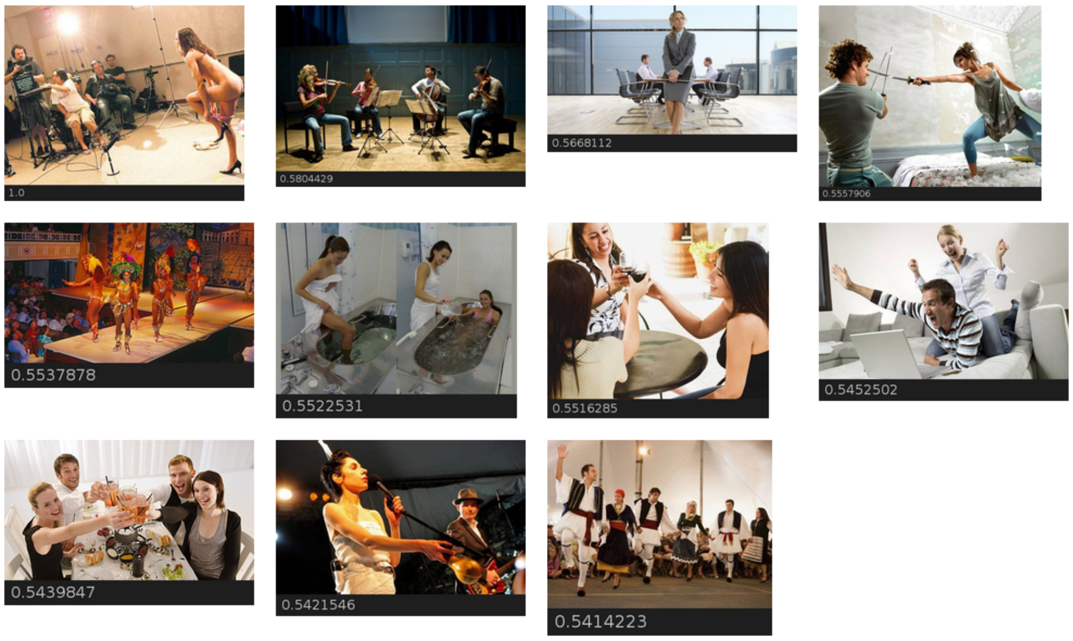


Figure 7.27: Original 4096 features + c_i scaling + $tanh$ sigmoid

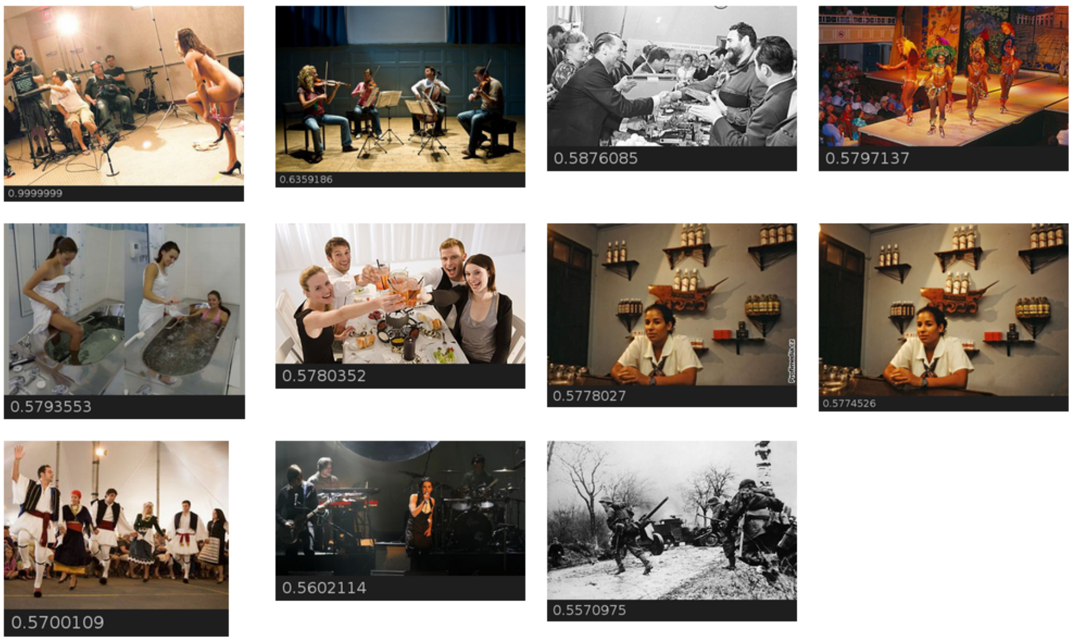


Figure 7.28: Denoising autoencoder, 1000 neurons

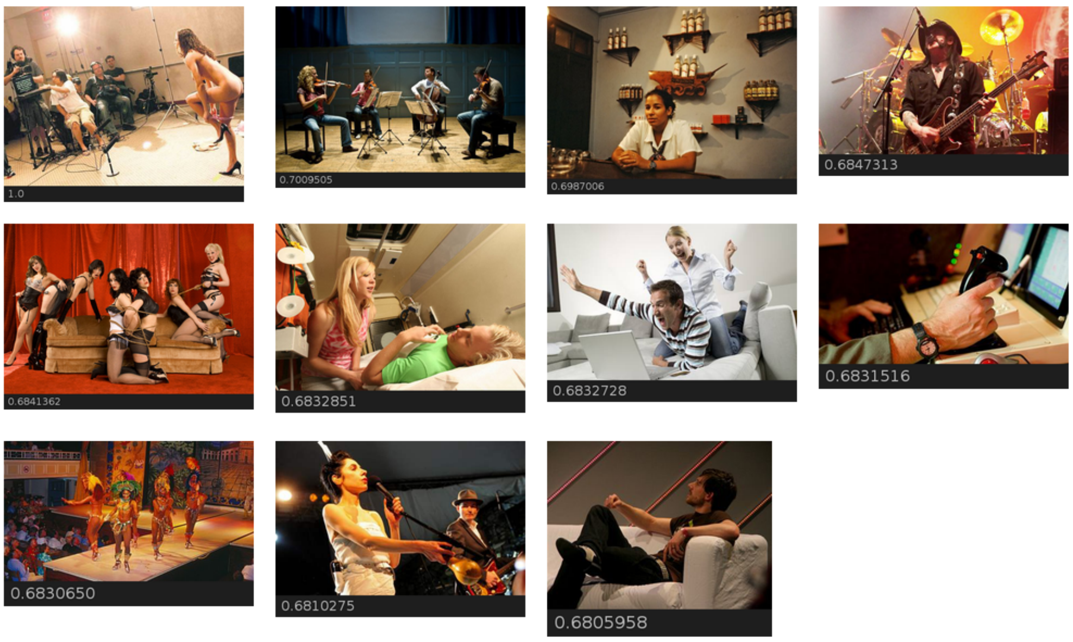


Figure 7.29: Stacked Denoising Autoencoder, 1000 + 250 neurons

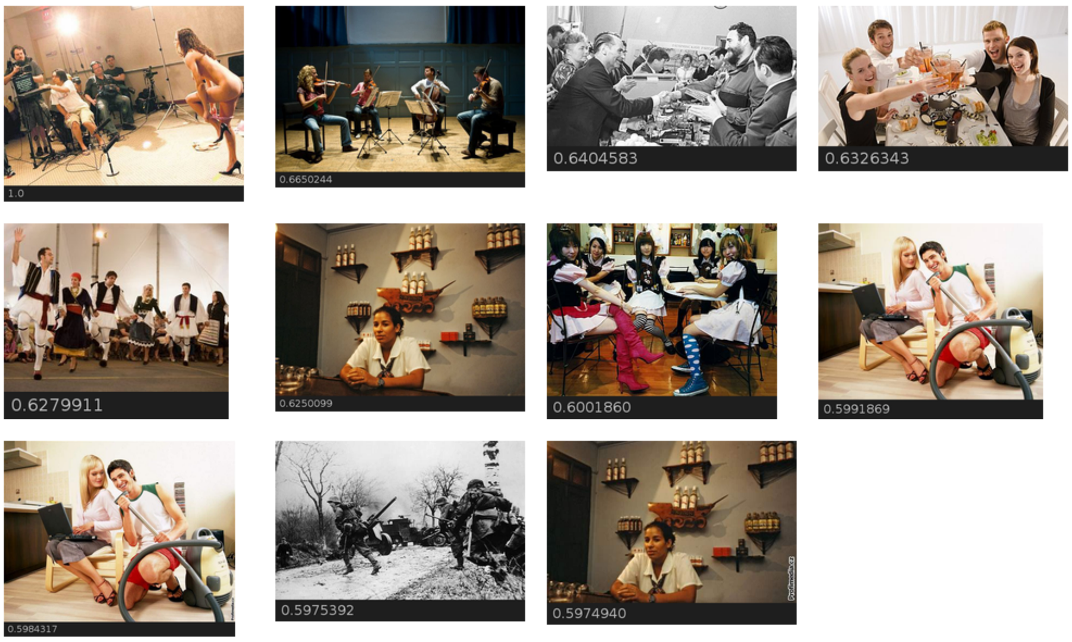


Figure 7.30: Restricted Boltzmann Machine, 1000 neurons

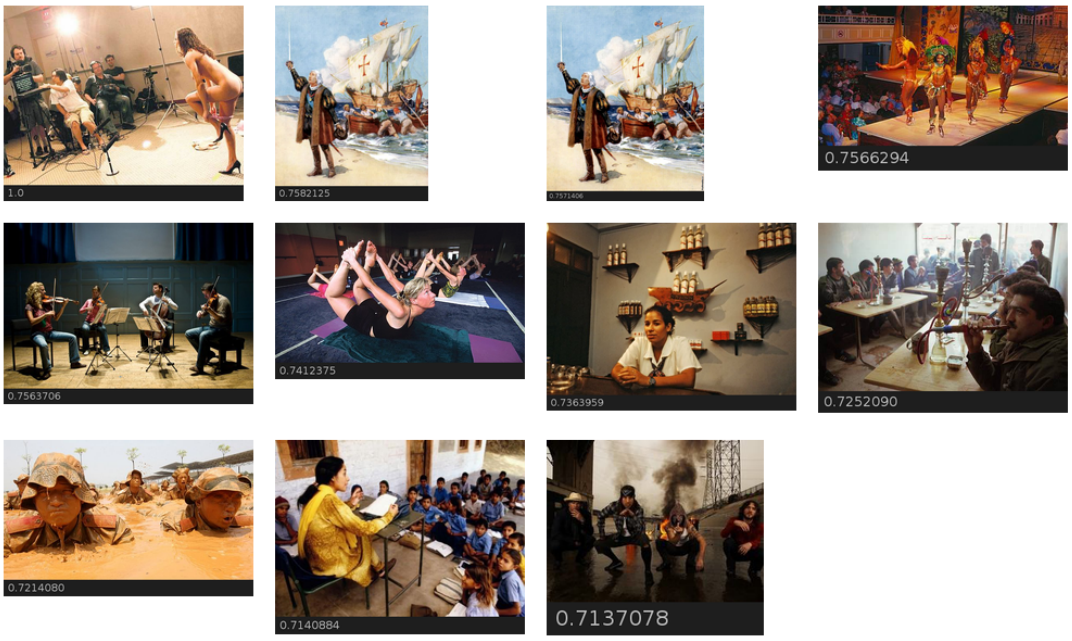


Figure 7.31: Stacked Restricted Boltzmann Machines, 1000 + 250 features

7.2.2 Visualizing learned representations

An important clue that the data is to “blame” for the failure of the text stack is the progression of several epochs in the beginning of training the 1000-neuron RBM. We plot the heatmap of the learned representations on a batch heldout data after 3 (Fig. 7.32), 6 (Fig. 7.33) and 30 (Fig. 7.33) epochs of training.

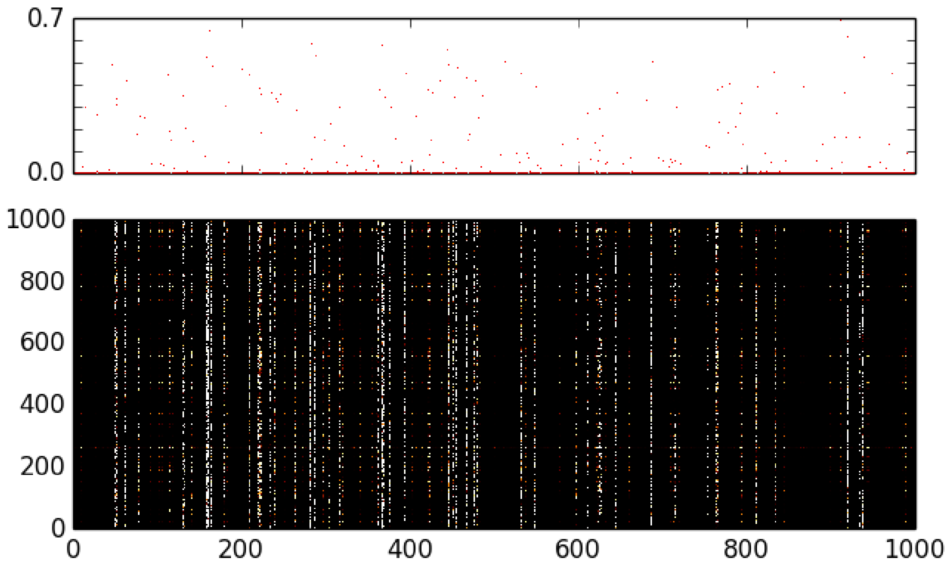


Figure 7.32: Representation learned with the 1000-neuron RBM, sigmoid activations, after the third epoch

We see that while features were quite correlated in the beginning, the learning successfully de-correlated them. This gives us certain hope that perhaps the trouble with the text modality was the result of an underestimated amount of time needed to train them, rather than a flaw in the models themselves.

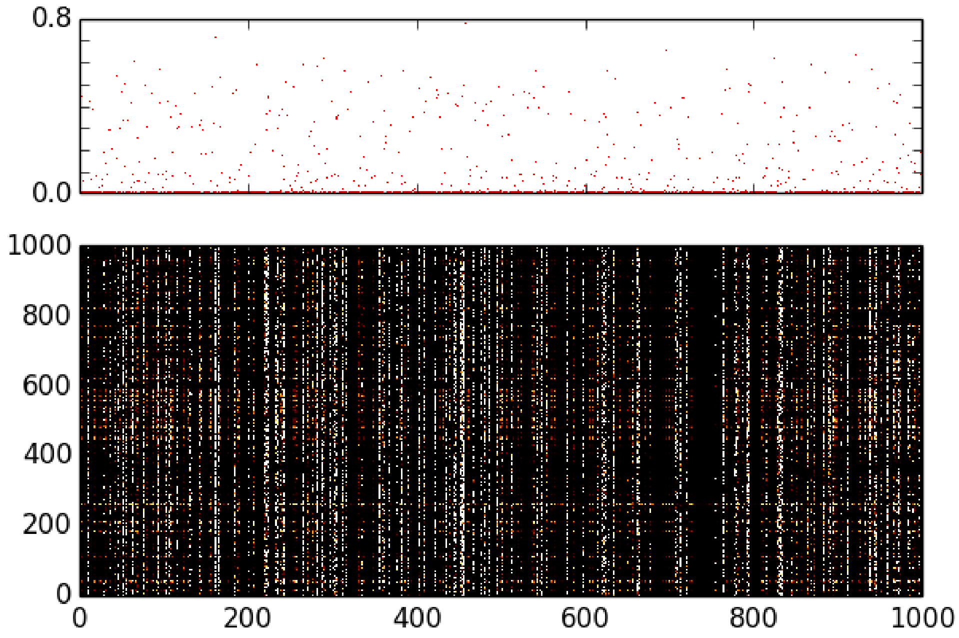


Figure 7.33: Representation learned with the 1000-neuron RBM, sigmoid activations, after the sixth epoch

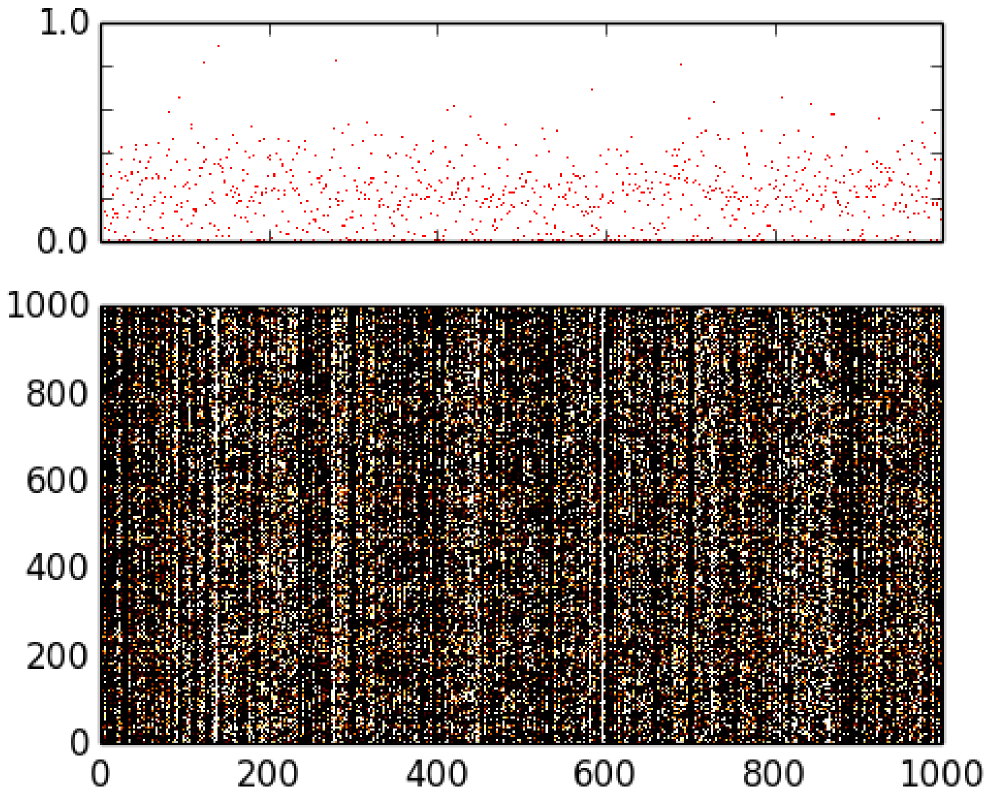


Figure 7.34: Representation learned with the 1000-neuron RBM, sigmoid activations, after epoch 30

8. Implementation

This chapter will shortly introduce tools available for deep learning and then give a high-level overview of our dedicated codebase, the Safire library.

Even on modern hardware, deep neural networks still require a lot of computing power. The algorithmic bottleneck for deep learning algorithms are the mathematical operations that happen inside neural networks. For this reason, it is highly beneficial to use a well-optimized mathematical library such as ATLAS [83], OpenBLAS [63] or Intel's MKL [19]. These libraries are written in C or C++ and implement the BLAS specification.[1]

Notably, for the Python programming environment, the NumPy [81] and SciPy [41] packages are available, which implement BLAS and can provide optimized math operations using these mathematical libraries. For Windows, the Entought Python Distribution (recently renamed Canopy) [1] is linked to an optimized version of MKL, which gives good speedups over a "plain" NumPy installation.[2]

8.1 Available tools for Deep Learning

Several neural network libraries exist that provide an implementation of the underlying algorithms and can utilize the fast mathematical libraries. Among these are FANN (Fast Artificial Neural Networks) [61], Caffe [40], Theano, [8] which we used, PyLearn2 [31] or Torch7 [17] (which is implemented over the somewhat exotic Lua language).

Furthermore, some of these libraries allow to transparently use CUDA GPUs [60], when available, which means practically no time has to be spent on GPGPU programming itself. For Python, which we used in our implementation, the cudamat library [56] is available for interfacing with CUDA code and the Theano library allows CUDA GPU use completely transparently with a configuration option (albeit only for 32-bit floats so far). In C++, the Caffe library also provides CUDA implementations. OpenCL [76] support among available tools is still minimal.

A remarkably elegant tool that deals with machine learning for NLP is the *gensim* Python library [64]. The library is designed to form *pipelines* from corpus, transformer and index objects. Corpora serve to feed data in the form of sparse vectors, transformers encapsulate machine learning algorithms as operations on sparse vectors or entire corpora and indexes are used for retrieval based on cosine similarity. The library is designed to keep a constant memory footprint using the *generator* construct of Python. We implemented our library on top of gensim's interfaces, wrapping our models as transformer objects.

[1] `https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms`

[2] Of course, on Linux systems, it would be much easier than on Windows to compile and optimize ATLAS, for a similar performance range.

8.2 Theano

For the computationally intensive part of our library, we settled on using the Theano library [8].[3] Programming with Theano comes in two stages. First, the programmer defines a *symbolic graph* of the computation. In the second stage, the symbolic graph is compiled into a C function. This second step is performed automatically - Theano parses the symbolic graph, applies numerous speed and stability optimizations, dynamically generates the C code and compiles it.

A notable feature is the `grad` operation, which automatically builds a symbolic graph for the *gradient* of another symbolic graph. This allows to define complex computations without worrying about how to differentiate them.

Theano has several distinct advantages:

- Transparent CUDA GPU integration
- Symbolic differentiation (no need to compute gradients manually)
- Automatic optimizations to improve speed and numerical stability
- Easy to implement various formulae
- Relatively mature project, good support, documentation and tutorials
- Multi-platform: both Windows and Linux

It also has several disadvantages:

- A *steep* learning curve (it takes considerable time to learn to think in symoblic expressions)
- Not quite perfect documentation
- Debugging is complicated, although well-documented
- Programming constructs like conditions or loops are difficult in symbolic expressions
- While possible to set up on 64-bit Windows, it is not easy (again, documentation is available)

Theano doesn't directly implement neural networks. However, its authors from the LISA lab[4] also provide a step-by-step tutorial on how to implement them using Theano.[5]

We chose Theano for implementing the mathematical "guts" of the models because of its performance, transparent use of GPU, symbolic differentiation and importantly solid documentation, support and an active community.[6]

8.3 The SAFIRE library

The SAFIRE library is the codebase for this work (and related future work). It is written in Python 2.7 using the EPD/Canopy distribution for 64-bit Windows. Besides packages for scientific computing that are a part of Canopy (numpy,

[3]`http://deeplearning.net/software/theano/introduction.html`

[4]`http://lisa.iro.umontreal.ca/index_en.html`

[5]`http://www.deeplearning.net/tutorial/`

[6]The theano-users Google group sees around 20 posts daily, with the library developers usually responding within hours. `https://groups.google.com/forum/#!forum/theano-users`

scipy and matplotlib), SAFIRE depends also on the gensim [64] and Theano [8] libraries.

The library is a medium-sized project, currently with about 24 000 lines of code. It is divided into the `safire` package, which contains the library, and a suite of scripts for preprocessing, training, running, evaluating, dataset manipulation, etc. that are built above the `safire` package. The scripts are designed so that they can be *also* used as pipeline components in an experiment: the functionality of each script is contained in a `main(args)` method where `args` is a namespace of command-line arguments and a `_build_argument_parser()` method that constructs the argument namespace and awaits initialization through Python's versatile `argparse` standard library mechanism. This way, any of the scripts can be called from within another Python module, enabling any users to construct their own wrappers around SAFIRE functionality smoothly, without resorting to `Subprocess.popen()` or `fork` calls.

8.3.1 Library design

The library is built around the gensim philosphy of *pipelines* that start with a source of data and feed vectors through various transformations, keeping to a low memory footprint. For retrieval, at the end of the pipeline, gensim's `Similarity` class is used as a search index.

The neural networks themselves are only loosely coupled to the gensim interfaces through a `SafireTransformer` wrapper class. The mechanism for training neural networks is somewhat of a library within a library, in the *safire.learning* module. It comprises of *models*, *model handles*, *updaters*, *learners* and a wrapper that implements the gensim `TransformationABC` interface, so that the trained models can be plugged into pipelines.

The model classes do not *do* anything; they implement *definitions* of their namesakes described in this work as Theano function graphs. These function graphs are constructed on model instance construction. In order to use the model – train it, use it to transform input data, etc. – *model handles* that provide an interface for training, validating and running the model. The same model can be re-used for different modes of operation by attaching to it multiple different handles: for instance, the joint layer of a multimodal model has a different handle for obtaining the joint representation from multimodal inputs and for sampling image features given clamped text features. Model handles are given to `SafireTransformer`s at initialization; for transforming an incoming data vector, the transformer only calls the handle's `run` method.

The models are initialized indirectly, using a `setup` class method. This method correctly initializes the model dimension and compiles the training, validation, test and run functions. This is a somewhat convoluted setup, which is brought about by the necessity to accomodate the idiosyncracies of theano: the building of the symbolic computational graph for the training and other functionalitites of the model is coupled to parameters of the learning algorithm such as batch size or learning rate. (Refactoring to reduce this coupling is planned.)

The training itself is also "outsourced" to classes separate from the model. Two classes participate in training: a *learner*, which runs the training, monitors progress, etc., and an *updater*, which defines the learning rule. Different learning algorithms as described in 3.3 would be implemented in updaters. Different learning strategies (like early-stopping) or facilities for monitoring progress are available in the learner. Also, saving progress and the capability to resume learning where it left off is implemented, although only in a very rudimentary form.

For feeding data to models and keeping a constant memory footprint at the same time, in line with the philosophy of gensim, a `ShardedDataset` class is available that feeds matrices of input data to the models and in the background dynamically loads *shards* from serialized files. (This action is modeled after the `Similarity` class is implemented in gensim.)

Facilities are also available for *experiment management*. A directory structure and naming scheme for keeping data and various components of the machine learning pipelines is defined in the `DataDirLayout` class, above which *Loaders* operate to feed the components to scripts that train, run or otherwise use them.

All components of the Safire library have save/load funcitonality. Due to restrictions on serialization using Python's `cPickle` module, a separate serialization scheme from gensim's `SaveLoad` class is employed for the model classes and classes that contain a model as a member which relies on taking a "snapshot" of serializable instance attributes that sufficiently characterize the instance. Each (sub)class with this save/load mechanism can be initialized using this instance attribute snapshot.

The `run.py` and `img_index_explorer.py` have an *interactive mode*, a very simple command-line interface, to explore the retrieval quality and the action of the multimodal pipeline. These capabilities are rudimentary at best; however, they have proven very useful and we plan to expand them significantly.

8.3.2 Available deep learning functionality

Classes for models belong into the `safire.learning.models` namespace. Models do not *do* anything; they implement *definitions*. The following models are implemented:

- Feedforward layer,
- Logistic regression classifier layer,
- Autoencoders,
- Denoising Autoencoders, Sparse Autoencoders,
- Restricted Boltzmann Machines (with sparsity and weight decay penalties as parameters),
- Replicated Softmax,[7]

The models can be concatenated together into a feedforward multilayer network, the `MultilayerPerceptron` class.

Only the updater for the standard SGD learning rule $\theta^{(k+1)} = \theta^{(k)} - lr * \nabla\theta$ is implemented.

[7]Known bug in sampling; in progress.

8.3.3 Advantages of the SAFIRE library

Once one is familiar with the Theano library,[8] it becomes easy to add new models. (Implementing a sparse denoising autoencoder when a denoising autoencoder was done was a matter of 20 minutes, including debugging.) This is thanks to Theano's automatic differentiation: within a class of models, only the loss function needs to be re-defined (and the corresponding parameters, like the sparsity target, have to be added to the initialization parameters and to the instance attribute snapshot, for persistence) and Theano takes care of the rest. In the same way, adding different optimization algorithms is a matter of implementing an `Updater` subclass (although it is necessary to be careful about what exactly needs to be updated *outside* the model as well, such as the matrix of previous updates for Rprop-family algorithms) and no modifications need to be made to the model *or* the `Learner` class that executes the training loop.

Care is taken to make data loading efficient and as little a bottleneck as possible, while maintaining a constant memory footprint. The *ShardedDataset* class allows for matrices to be accessed in a way that the memory footprint stays constant[9] by splitting the entire dataset into *shards* on disk and reading only the current shard into memory. For sequential access, which is usually the case when training a network, this works very well; for non-sequential access such as in the `ShardedMultimodalDataset` class, a simple cache is implemented to alleviate the effects of "shard jumping".

We also attempted to make error messages as informative as possible and include a command-line option to training scripts for debugging the *insides* of Theano-compiled functions. In addition, each script has a standard `-v` and `--debug` interface for turning on lower levels of progress logging. (It is recommended to use the `--verbose`, or `-v`, option, as it provides the best balance between being able to monitor that everything is running OK and having a *lot* of clutter in your console that is useless unless actually debugging an error.)

8.3.4 Disadvantages and future work

Although we have been able to carry out various experiments and the SAFIRE proved useful, flexible and efficient, it is not yet a mature software product from a software engineering perspective. During extensive use for this work, it has grown to a point where refactoring into a separate experiment management and machine learning library would be beneficial.

The more pressing issues from the software development cycle point of view are:

- Some unit tests are available but mostly re-purposed for scenario testing to verify that the pipelines for basic training and runtime tasks work. Unit tests in the true sense of the word are available only for a small number of utility functions.

[8] Which is not easy, but quite worth familiarizing oneself with, anyway.

[9] With the caveat of a known memory leak that is the result of some strange interaction of the "self" special instance method argument with Python 2.7's garbage collector in a special case of indexing the dataset.

- While API documentation is generated using the Sphinx [12] `autodoc` module and – at least for the deep learning components – relatively complete, user documentation is available only in the form of a tutorial and introductory remarks to individual classes with doctest examples.
- Facilities for distributing the library using the standard `distutils` library – the ubiquitous `setup.py` script – is not available.

Features missing, incomplete or in need of redesign:

- Facilities for monitoring the learning process, while available, need to be refactored away to a separate Monitor class that the learner uses.
- The experiment management component needs to be able to automatically and consistently generate labels for experiment components based on input parameters.
- Coupling between the learning process parameters, gradient descent updates and model at model initialization complicate library design. While the `Updater` classes are a step in this direction, it should be investigated how to de-couple them further.
- The process of setting up training updates for Contrastive Divergence training in RBMs needs to be refactored, as there is currently a coupling that is preventing delegating the update step generation to an `Updater` class.

8.4 The attached CD

The Safire library with a sample data set and tutorial is included with this thesis as supplementary material on a CD enclosed with this thesis. There is a `README.rst` text file in the root directory that contains all necessary information on running the library.

9. Conclusions and Discussion

This work has not attained its main goal. We were unable to train a working deep learning model for the text-image modeling task. This is due to several factors:

- **Difficulty of training deep learning models.** We underestimated how complex the training process for a multi-layer unsupervised neural network is.
- **Strategic miscalculations**, which are detailed below. The strategic errors made caused a drain on resources – time – that accounts for the lack of positive results.

First, the difficulty of training a deep neural network was severely underestimated.

Second, while the SAFIRE library is not a bad tool per se, it was a much greater development project than anticipated. Programming took up far too much time and not enough time was left for experimentation. While the volume of code (roughly 24 000 lines) may sound like a significant amount of work and the library is ready for running and managing experiments, it is still not near a mature software product. The decision to create a dedicated codebase for both the experiment management and machine learning parts (although this division only became clear as the specification evolved) was probably wrong; the available tools like the pylearn2 library should have been investigated more thoroughly.

Also, further time was spent on coordinating the annotation project which, while promising for the research of article-illustrative image relationships, has had little impact on the experiments.

In general, the work is at a sort of half-way point. The Safire library, while not a mature piece of software, can be used for experimenting in earnest and is easily extensible, and the image features obtained from the ImageNet CNN are appropriate for content-based retrieval, However, as evidenced by the lack of progress, more experimentation is needed on the text document models. We also need a greater understanding of how the networks train and how to interpret different types of failures to be able to design a model and set hyperparameters so that learning doesn't converge to a degenerate solution.

9.1 Future work

First, there is some hope with regard to overcoming the sparsity issue: based on the observations in Sect. 7.1, we tried training a 250-neuron RBM on the text data with L2 weight decay and extra bias decay and a vastly prolonged training schedule (1000s of epochs, after the observations in Sect. 7.2.2). The learned representations on a heldout sample eventually looked like (Fig. 9.1). While strong correlations between feature activities are still pronounced, the model has been able to move away somewhat from the $P(x_i \mid \mathcal{D})$ loss function plateau.

It is possible that if this process was run for several more days or (probably rather) weeks, a useful representation could indeed have been learned.

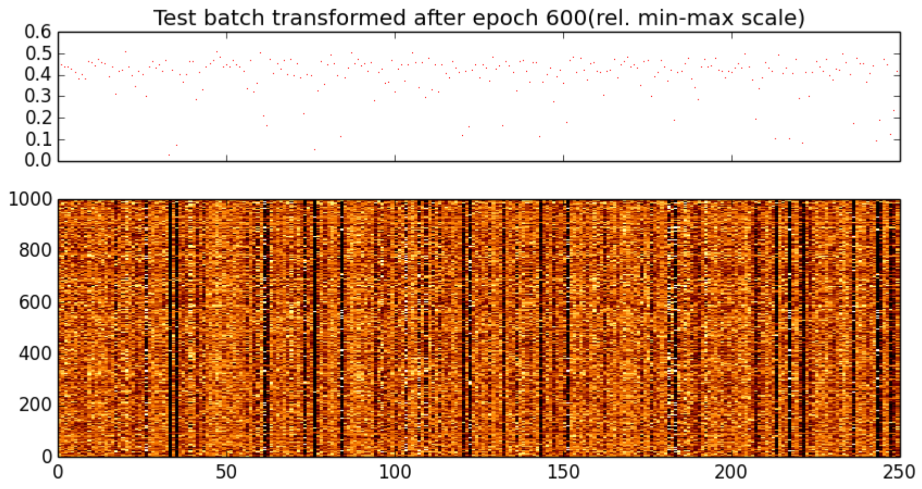


Figure 9.1: 250-neuron RBM on normalized data with weight decay

More optimization methods such as momentum, rprop or second-order methods should also be explored. Also, dropout and other types of noise than zero-masking should be applied, and better normalization of data such as the centering trick for RBMs should be applied.

We plan to investigate priors for the learned representations with more complex desirable patterns of activation; notably sparse Dirichlet or Pitman-Yor priors for multinomial units such as in the Replicated Softmax model. Some research has been done on Spike-and-Slab priors for RBMs, which also have interesting sparsity properties [20] [30].

On a different track, we plan to use neural network language models like the skip-grams of Mikolov et al. [54] as a transformation of the input, to gain a somewhat denser and less volatile representation for words. Training such models, however, will require orders of magnitude larger text data than we had.

While released as supplementary material for this thesis, much work remains on making the SAFIRE library ready for a full release. The merit of maintaining the library and its components will have to be assessed.

The manual annotation of the dataset will be further investigated in order to understand the relationship we are trying to model. A better understanding here may lead to more suitable text preprocessing steps.

9.2 Concluding remarks

It may well be that the image (Fig. 9.2) that was retrieved by the failed multimodal model referenced in 7 as the top candidate for over 60 % of texts (and was second or third for practically all others) is exactly the image worth these previous (several) thousand words.

While we have produced a resoundingly negative result on training the text modality and were unable to build a working multimodal model, we are hopeful that the lessons learned will help us to train better models in the future.

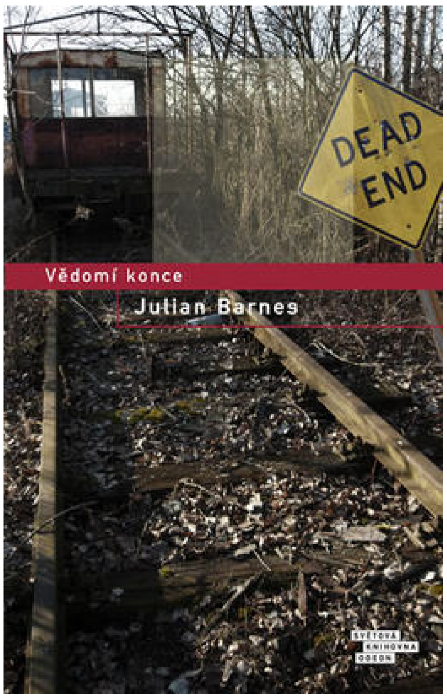


Figure 9.2: The image that was retrieved as the top candidate for over 60 percent of the evaluation data texts. A fitting conclusion? Let us hope not entirely.

Bibliography

[1] Enthought canopy (version 1.4.0), 2014.

[2] Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[3] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[4] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[5] Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Comput.*, 21(6):1601–1621, June 2009.

[6] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Université De Montréal, and Montréal Québec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[7] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *CoRR*, abs/1305.6663, 2013.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010. Oral Presentation.

[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[10] Léon Bottou. Stochastic gradient tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

[11] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[12] Georg Brandl et al. Sphinx, 2007.

[13] Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Rates-fpcd: A better-mixing sampling procedure for rbms. 2010.

[14] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning. 2005.

[15] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[16] Adam Coates and Andrew Ng. The importance of encoding versus training with sparse coding and vector quantization. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 921–928, New York, NY, USA, June 2011. ACM.

[17] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. 2011.

[18] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[19] Intel corporation. Math kernel library, January 2014.

[20] Aaron C. Courville, James Bergstra, and Yoshua Bengio. A spike and slab restricted boltzmann machine. In *AISTATS*, pages 233–241, 2011.

[21] George E. Dahl, Ryan Prescott Adams, and Hugo Larochelle. Training restricted boltzmann machines on word observations. *CoRR*, abs/1202.5695, 2012.

[22] Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered Markov Chain Monte Carlo for training of restricted Boltzmann machines. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, May 13-15, 2010, Chia Laguna Resort, Sardinia, Italy*, pages 145–152, 2010.

[23] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[24] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, March 2010.

[25] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.

[26] Yansong Feng and Mirella Lapata. Topic models for image annotation and text illustration. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 831–839, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. `http://dl.acm.org/citation.cfm?id=1857999.1858124`.

[27] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[28] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[30] Ian J. Goodfellow, Aaron C. Courville, and Yoshua Bengio. Spike-and-slab sparse coding for unsupervised feature discovery. *CoRR*, abs/1201.3382, 2012.

[31] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[32] Jan Hajič. *Disambiguation of Rich Inflection: Computational Morphology of Czech.* Karolinum, nakladatelstvi Univerzity Karlovy, 2004.

[33] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[34] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[35] Geoffrey Hinton and Ruslan Salakhutdinov. Discovering binary codes for documents by learning deep generative models. *Topics in Cognitive Science*, 3(1):74–91, 2011.

[36] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[37] Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. In *Proceedings of the 27 th International Confer- ence on Machine Learning*, 2010.

[38] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade (2nd ed.)*, pages 599–619. 2012.

[39] Christian Igel and Michael Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50(0):105 – 123, 2003.

[40] Yangqing Jia. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013.

[41] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2014-07-19].

[42] Özgür Kişi and Erdal Uncuoğlu. Comparison of three back-propagation training algorithms for two case studies. *Indian Journal of Engineering and Materials Sciences (IJEMS)*, pages 434–442, October 2005.

[43] Ryan Kiros, Ruslan Salakhutdinov, and Zemel. Multimodal neural language models. In *Journal of Machine Learning Research: Workshop and Conference Proceedings*, volume 32, pages 595–603, 2014.

[44] A. E. Kostopoulos and T. N. Grapsa. Self-scaled conjugate gradient training algorithms. *Neurocomputing*, (72):3000–3019, 2009.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[46] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 536–543, New York, NY, USA, 2008. ACM.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[48] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[49] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168.

[50] University of Montreal LISA lab. Deep learning tutorials, 2008.

[51] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013.

[52] James Martens. Deep learning via hessian-free optimization. In *ICML*, pages 735–742, 2010.

[53] James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade (2nd ed.)*, pages 479–535. 2012.

[54] Tomáš Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[55] Tomáš Mikolov, Stefan Kombrink, Anoop Deoras, Lukáš Burget, and Jan Černocký. Rnnlm - recurrent neural network language modeling toolkit. In *Proceedings of ASRU 2011*, pages 1–4. IEEE Signal Processing Society, 2011.

[56] Volodymyr Mnih. Cudamat: a cuda-based matrix class for python. Technical report, 2009.

[57] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[58] Andrew Ng. Sparse autoencoders.

[59] Jiquan Ngiam, Zhenghao Chen, Sonia A. Bhaskar, Pang W. Koh, and Andrew Y. Ng. Sparse filtering. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1125–1133. Curran Associates, Inc., 2011. `http://papers.nips.cc/paper/4334-sparse-filtering.pdf`.

[60] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.

[61] S. Nissen. Implementation of a fast artificial neural network library (fann). Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003. http://fann.sf.net.

[62] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *CoRR*, abs/1406.1831, 2014.

[63] Wang Qian, Zhang Xianyi, Zhang Yunquan, and Qing Yi. Augem: Automatically generate high performance dense linear algebra kernels on x86 cpus. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, Denver, CO, November 2013.

[64] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[65] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994.

[66] M. Riedmiller. Rprop - description and implementation details. Technical report, 1994.

[67] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, 1993.

[68] M Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[70] Ruslan Salakhutdinov and Geoffrey E. Hinton. Replicated softmax: an undirected topic model. In *NIPS*, pages 1607–1614, 2009.

[71] Tom Schaul, Sixin Zhang, and Yann LeCun. No More Pesky Learning Rates. In *International Conference on Machine Learning (ICML)*, 2013.

[72] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, pages 129–136, 2011.

[73] N. Srivastava and R. Salkhutdinov. Multimodal learning with deep boltzmann machines. *Proceedings of the 2012 NIPS conference*, 2012.

[74] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[75] Nitish Srivastava, Ruslan Salakhutdinov, and Geoffrey E. Hinton. Modeling documents with deep boltzmann machines. *CoRR*, abs/1309.6865, 2013.

[76] John E. Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, May 2010.

[77] Milan Straka and Jana Straková. MorphoDiTa: Morphological dictionary and tagger, 2014.

[78] R. L. Stratonovich. Conditional markov processes. *Theory Probab. Appl.*, 5(2):156–178, 1960.

[79] T. Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM New York, NY, USA, 2008.

[80] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[81] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, March 2011.

[82] Nan Wang, Jan Melchior, and Laurenz Wiskott. Gaussian-binary restricted boltzmann machines on modeling natural image statistics. *CoRR*, abs/1401.5900, 2014.

[83] R. Clint Whaley and Antoine Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

[84] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

List of Tables

List of Figures

Attachments

Deriving the gradient of RBM negative log-likelihood

Given an energy function E and the following definitions of the free energy $\mathcal{F}$, probability distribution $P(x)$ and normalization constant Z:

$$\mathcal{F}(x) = -log \sum_h e^{-E(x,h)} \tag{9.1}$$

$$Z = \sum_{\tilde{x}} e^{-\mathcal{F}(\tilde{v})} \tag{9.2}$$

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \tag{9.3}$$

we obtain the following gradient of the negative log likelihood $\ell(x)$ with respect to the model parameters θ:

$$\begin{aligned}
-\frac{\partial logp(x)}{\partial \theta} &= \left[-log\frac{e^{-\mathcal{F}(x)}}{Z}\right]'_\theta \\
&= -\left[loge^{\mathcal{F}(x)} - logZ\right]'_\theta \\
&= \frac{\partial \mathcal{F}(x)}{\partial \theta} - [logZ]'_\theta
\end{aligned} \tag{9.4}$$

Computing $\frac{\partial logZ}{\partial \theta}$:

$$\begin{aligned}
\frac{\partial logZ}{\partial \theta} &= \frac{1}{Z}\frac{\partial Z}{\partial \theta} \\
&= \frac{1}{Z}\left[\sum_{\tilde{x}} e^{-\mathcal{F}(\tilde{x})}\right]'_\theta \\
&= -\sum_{\tilde{x}} \frac{e^{-\mathcal{F}(\tilde{x})}}{Z}\frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta} \\
&= -\sum_{\tilde{x}} P(\tilde{x})\frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}
\end{aligned} \tag{9.5}$$

and plugging into 9.4 :

$$-\frac{\partial logp(x)}{\partial \theta} = \frac{\partial \mathcal{F}(x)}{\partial \theta} - \sum_{\tilde{x}} P(\tilde{x})\frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta} \tag{9.6}$$