Lappeenranta University of Technology
Faculty of Technology Management
Degree Programme in Computer Science
Major in Intelligent Computing

Master's Thesis

Mikko Haavisto

PRETRAINING CONVOLUTIONAL NEURAL NETWORKS FOR VISUAL RECOGNITION

Examiners: Professor Lasse Lensu
Associate Professor Arto Kaarna

Supervisor: Professor Lasse Lensu
Associate Professor Arto Kaarna

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Programme in Computer Science
Major in Intelligent Computing

Mikko Haavisto

Pretraining Convolutional Neural Networks for Visual Recognition

Master's Thesis

2016

56 pages, 18 figures and 4 tables.

Examiners: Professor Lasse Lensu
Associate Professor Arto Kaarna

Keywords: deep learning, convolutional neural network, pretraining

Convolutional Neural Networks (CNN) have become the state-of-the-art methods on many large scale visual recognition tasks. For a lot of practical applications, CNN architectures have a restrictive requirement: A huge amount of labeled data are needed for training. The idea of generative pretraining is to obtain initial weights of the network by training the network in a completely unsupervised way and then fine-tune the weights for the task at hand using supervised learning. In this thesis, a general introduction to Deep Neural Networks and algorithms are given and these methods are applied to classification tasks of handwritten digits and natural images for developing unsupervised feature learning. The goal of this thesis is to find out if the effect of pretraining is damped by recent practical advances in optimization and regularization of CNN. The experimental results show that pretraining is still a substantial regularizer, however, not a necessary step in training Convolutional Neural Networks with rectified activations. On handwritten digits, the proposed pretraining model achieved a classification accuracy comparable to the state-of-the-art methods.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma
Älykkään laskennan pääaine

Mikko Haavisto

Konvoluutioneuroverkkojen esiopettaminen visuaaliseen tunnistamiseen

Diplomityö

2016

56 sivua, 18 kuvaa ja 4 taulukkoa.

Tarkastajat: Professori Lasse Lensu
Tutkijaopettaja Arto Kaarna

Hakusanat: syvä oppiminen, konvoluutioneuroverkko, esiopetus
Keywords: deep learning, convolutional neural network, pretraining

Konvoluutioneuroverkoista (KNV) on tullut viimeisintä tekniikkaa edustavia menetelmiä laajamittaiseen visuaaliseen tunnistamiseen. Moniin käytännön sovelluksiin KNV arkkitehtuureissa on rajoittava vaatimus: Ne tarvitsevat koulutusta varten laajan joukon luokiteltua dataa. Generatiivisen esiopetuksen ideana on saada verkon alustavat painot ohjaamattoman koulutuksen avulla ja tämän jälkeen hienosäätää painoja ohjatusti tiettyä tehtävää varten. Tässä työssä esitellään syviä neuroverkkoja ja algoritmeja ja näitä menetelmiä sovelletaan käsinkirjoitettujen numeroiden ja luonnollisten kuvien luokittelutehtäviin. Työn tavoitteena on selvittää vaimentaako KNV:n viimeaikainen optimointi- ja regularisointimenetelmien kehitys esiopetuksen vaikutusta. Kokeellisten tulosten perusteella esiopetus on yhä merkittävä regularisointimenetelmä, mutta ei kuitenkaan välttämätön vaihe konvoluutioneuroverkkojen opetuksessa. Ehdotettu esiopetusmalli saavutti luokittelutarkkuuden, joka on käsinkirjoitetuissa numeroissa verrannollinen alan parhaimpiin menetelmiin.

PREFACE

I wish to thank my supervisors and examiners Professor Lasse Lensu and Associate Professor Arto Kaarna. Also, I would like to thank the Machine Vision and Pattern Recognition Research Laboratory in Lappeenranta University of Technology for funding this project.

Finally, I would like to thank Anu Hiltunen for the support and understanding during this work.

Lappeenranta, April 20th, 2016

Mikko Haavisto

CONTENTS

ABBREVIATIONS

CBIR	Content Based Image Retrieval
CD	Contrastive Divergence
CDBN	Convolutional Deep Belief Net
CE	Cross-Entropy
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
DBN	Deep Belief Net
DNN	Deep Neural Network
DSM	Denoising Score Matching
GD	Gradient Descent
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
MSE	Mean Squared Error
NAG	Nesterov's Accelerated Gradient
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Units
RGB	Red, Green, Blue
SGD	Stochastic Gradient Descent
SM	Score Matching
SVM	Support Vector Machine
ZCA	Zero-phase Component Analysis

SYMBOLS

$\boldsymbol{a}$	Hidden unit's bias vector
$\boldsymbol{b}$	Visible unit's bias vector
E	Energy function
$E_{P(x)}\left[\cdot\right]$	Expectation with respect to probability distribution P
g	Activation function
$\boldsymbol{h}$	Hidden (latent) units vector
$\boldsymbol{h^k}$	k-th feature map
$\mathcal{N}$	Normal distribution
P	Probability distribution function
q_σ	Corruptor model: isotropic Gaussian of variance σ^2
$\boldsymbol{v}$	Visible (observed) units vector
$\tilde{\boldsymbol{v}}$	Corrupted visible units vector
$\hat{\boldsymbol{v}}$	All possible configurations of visible units
$\boldsymbol{W}$	Weight matrix
$\boldsymbol{W}^k$	k-th feature detector
$\tilde{\boldsymbol{W}^k}$	Flipped k-th feature detector
$\boldsymbol{x}$	Input vector
α	Learning rate
ψ	Score function
σ	Standard deviation
θ	Model parameters
$*$	Convolution operation
$\nabla_\theta f$	Gradient of f with respect to θ
μ	Momentum coefficient

1 INTRODUCTION

1.1 Background

Training deep computational models that can learn representations of data with multiple levels of abstractions has been of great interest to computer vision researchers. A depth refers to the number of levels of composed non-linear operations in the learned function. Most current supervised learning algorithms correspond to shallow architectures (1, 2 or 3 levels), such as neural networks with one hidden layer, Support Vector Machines (SVMs) and random forests. One way to achieve this goal is to use a Deep Neural Network (DNN) architecture. Neural networks are by no means a recent innovation and the challenges of its optimization algorithm, called backpropagation, are widely known in the machine learning community. Backpropagation is a supervised learning algorithm, which forms its learning rule by propagating output layer's error signal using the chain rule [1]. Since it uses the "hill climbing" strategy of gradient descent, which means that the weights are updated in batches, it can only converge to a local minimum. However, the real problem with the traditional neural network with sigmoid activation is that the solution is very sensitive to the initial conditions and often yields suboptimal solutions because of the optimization difficulties [2].

In 2006, Hinton et al. [3] introduced unsupervised learning algorithm which can be used to pretrain deep generative models such as Deep Belief Nets (DBNs). After unsupervised pretraining, a traditional feed-forward neural network is initialized using the weights from the pretrained model and the network is fine-tuned with backpropagation. This method, to successfully train deep models, was a huge breakthrough that eventually led to a new area of machine learning research called deep learning [4].

Since 2006, many developments in deep learning have been made. Most notably, researchers in the field have turned their interests back to methods of supervised learning. There are at least three reasons for this. First, the hardest problems in computer vision have interested people for a long time. Therefore, there are big labeled datasets available such as ImageNet [5] which contains millions of natural images. Second, better nonlinear activation functions for hidden units have been discovered. For example, Rectified Linear Units (ReLU) [6] preserve information about hidden units relative intensities and learn features that are better for object recognition. Finally, in 2012, Hinton et al. developed a regularization technique called dropout [7]. It is an interesting biology-inspired method to prevent overfitting by randomly omitting half of the feature detectors on each training

case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors.

Recently in 2013, Goodfellow et al. [8] developed Maxout Networks, which provided state-of-the-art classification performance on many commonly used research datasets. Maxout activation function is a generalization of rectified linear unit and it has beneficial characteristics both for optimization and model averaging with dropout.

An image can be reshaped to a vector to use it as an input for a fully-connected neural network architecture. This approach is sufficient for tiny images, but scaling fully-connected neural networks for realistic-sized images (e.g. 200×200 pixels) remains challenging for two reasons [9]:

1. When input dimensions increases, the number of the network parameters increases at a much higher rate.
2. The fully-connected architecture has no built-in invariance with respect to translations, or local distortions of the inputs. Objects can appear at arbitrary locations in images and a DNN needs to learn weights that detect a given feature separately for every location.

Convolutional Neural Network (CNN) tries to mitigate these problems by using three architectural ideas: local receptive fields, shared weights and spatial sub-sampling. With the local receptive fields, the first convolutional layer can detect simple low-level features such as edges or corners. These features are then combined by the subsequent convolutional layers in order to detect higher-order features. By convolving a feature detector, the weights of the detector are shared over all locations of the image. Spatial sub-sampling, also referred as pooling, shrinks the representation of the detection layer providing some degree of translational invariance [9].

A large CNN architecture, now commonly referred as AlexNet, was considerably better than the previous state-of-the-art methods in classification and localization tasks in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [10, 11]. The network contained 60 million parameters and 650,000 neurons and was trained on raw RGB pixel values. This was a major success and because of it, CNN architecture received a lot of attention in the field of object recognition and computer vision in general. ImageNet recognition challenge is an annual competition and the progress has also been remarkable after the success of AlexNet. In ILSVRC 2014 classification task [12], the

winner, GoogLeNet [13] achieved 8.66% decrease in the classification error compared to AlexNet's error. GoogLeNet architecture contained 5 million parameters, which is a result of using more convolutions instead of fully connected layers. It only had a modest increase, less than two times, in the number of operations compared to AlexNet. This suggest that the success of deep neural networks in large-scale vision problems, started in 2012, has potential to progress also in the near future.

Even though the recent progress in deep learning is driven by supervised learning methods, practical machine vision problems do not commonly have large labeled datasets. Thus, it is clear that methods that can benefit from unlabeled data are very important in practical applications. Some applications, like Content Based Image Retrieval (CBIR), is often purely unsupervised. Unsupervised models are also helpful in analyzing deep models and removing or tolerating occlusions because they try to model the input distribution, which means samples can be generated from the model. In 2009, Lee et al. [14] introduced convolutional DBNs that combines the benefits of unsupervised learning and scalable convolutional networks.

There is also one interesting aspect, why unsupervised generative models may become more practical in the future. A very recent study on the properties of deep neural networks constructed adversarial examples, which were misclassified by a deep network [15]. These adversarial examples are indistinguishable from regular examples by humans. This is in contradiction with the network's ability to achieve high generalization performance. Another study on this topic revealed that discriminative DNN models are easily fooled and showed that this does not apply only to the adversarial examples, but also to many other unrecognizable images [16]. Their leading hypothesis is that discriminative models create decision boundaries that partition the data into regions, and in high-dimensional spaces, an unrecognizable image can be far from both, the decision boundary and natural images in that area, causing the image to be classified with a high confidence. On the contrary, a generative model, such as DBN, builds a density model of the input images, corresponding to a much smaller area per class. This could be a vital property in obtaining a vision system, which cannot be as easily exploited.

1.2 Objectives and Restrictions

The objective of this thesis is to study whether pretraining is useful in training Convolutional Neural Networks for recognition tasks. The main goal is to find out if the effect of pretraining is damped by recent practical advances in optimization and regularization of

deep networks.

This thesis will focus on pretraining CNNs with Rectified Linear activations. ReLUs have widely replaced older sigmoid activations in the literature and applications. While it is practical and relatively easy to increase the performance by combining results of several networks, this kind of model averaging is excluded from the scope of the thesis.

1.3 Structure of the Thesis

This thesis is structured as follows. A detailed description of deep neural networks are given in Section 2. Section 3 introduces different regularization methods for DNNs, including generative pretraining, and gives overview of gradient-based optimization strategies. The experiments on visual recognition tasks are explained in Section 4. The results are discussed in Section 5 and a short conclusion of this thesis is given in Section 6.

2 DEEP NEURAL NETWORK

2.1 Feed-Forward Architecture

A hidden layer in a neural network corresponds to multiple small computing units. These units have two computational steps. First, a linear step that corresponds to a weighted sum of its inputs and an offset determined by the biases. The real power of a neural network lies in the second step, where a nonlinear function, also commonly referred as the activation, is applied. These small nonlinearities can be heavily composed by adding more layers to the network to approximate any function in a computationally efficient way [17].

In a feed-forward neural network architecture, a layer has a directed connection to the layer above it and such a network does not impose recurrent connections from top layers back to bottom layers [1]. The units do not have connections within a layer and this property makes the implementation of neural networks highly practical. Calculating the activations in a layer can be easily parallelized and given the computing power of todays Graphical Processing Units (GPU), it is feasible to train deeper networks.

The basic architecture of a DNN contains at least one hidden layer with a nonlinear transformation and one output layer, as illustrated in Figure 1. The data is fed into the first layer, which means that the first activations h_0 simply corresponds to the values of the data. The hidden layers act as learned feature extractors, and the purpose of the output layer is to tie the features to the particular task at hand. For example, different output layers are needed for classification and regression, but these problems may use similar architecture for the hidden layers. In a classification problem, a typical choice for an output layer is the softmax layer whereas a regression problem can simply utilize a linear layer.

2.2 Connectivity

2.2.1 Fully Connected Layer

The standard fully connected layer architecture, illustrated in Figure 2a, does not add topological priors on the problem to the network architecture [9]. It can be used when the input dimension is relatively small. It is suitable also for general cases, when one does

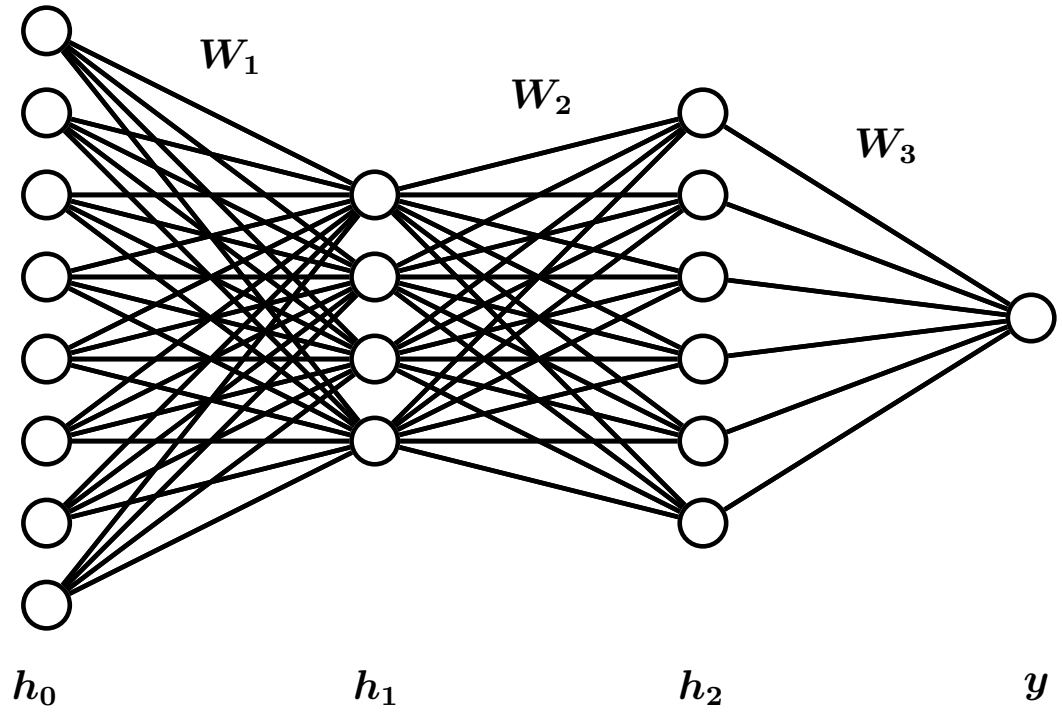


Figure 1. Deep Neural Network. The circles represent the units in a layer $\boldsymbol{h_i}$ and the edges correspond to the weights $\boldsymbol{W_i}$ between the layers $\boldsymbol{h_i}$ and $\boldsymbol{h_{i-1}}$. The final layer $\boldsymbol{y}$ corresponds to the output value of the network.

not have prior information on the problem or the task is to find out how well the network can learn particular structures without giving these priors.

The features $\boldsymbol{h}$ of a fully connected layer with X input units and H hidden units are obtained by:

$$\boldsymbol{h} = g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{1}$$

where $g(x)$ is an activation function, $\boldsymbol{W}^{H\times X}$ is the weight matrix, $\boldsymbol{b}^H$ is the bias vector and $\boldsymbol{x}^X$ is the input. In the fully connected architecture, a hidden unit corresponds to a global feature detector. On the contrary, image data essentially consist of highly local structures, which are composed to form higher level features. A fully connected architecture poses a problem, where local features correspond to a weight matrix with many zero elements. This is a very inefficient way to learn local structures and it is only suitable for tiny images.

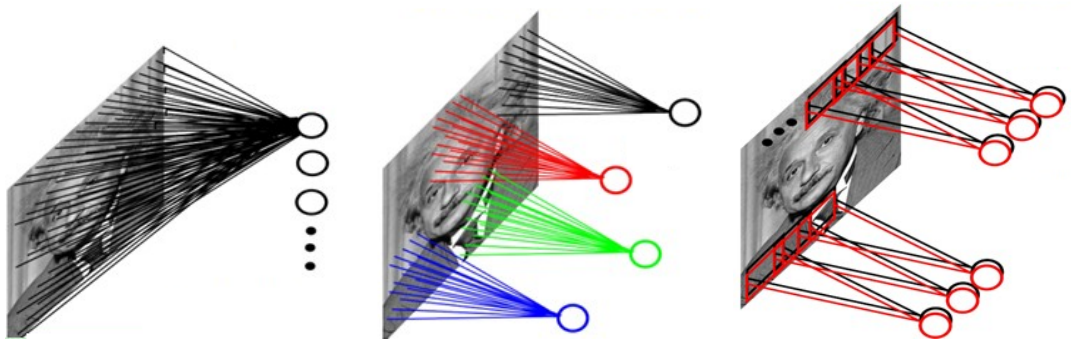
2.2.2 Convolutional Layer

One way to solve the scaling problem with the fully connected architecture is to simply use local connections, as illustrated in Figure 2b. This approach has one crucial problem: The feature detectors are tied to specific locations. To avoid replicated feature detectors it becomes obvious to share detectors across all locations of the image. This weight sharing principle for the locally connected architecture is exactly what a convolution does. A convolutional network architecture [9], illustrated in Figure 2c, makes it feasible in terms of memory requirements to scale up the dimensionality of the input and the number of feature detectors.

The convolution operation $*$ is also a linear operation, similar to the matrix multiplication in Eq. 1 in the case of the fully connected architecture. If we denote the k-th feature detector as $\boldsymbol{W}^k$ and the corresponding bias as b^k, the k-th feature map is obtained as follows:

$$\boldsymbol{h}^k = g(\boldsymbol{W}^k * \boldsymbol{x} + b_k). \tag{2}$$

A filter is another name for a feature detector using the analogy of convolution for image processing. If the boundaries of the image in the convolution are handled by a zero padding scheme, a feature map will have the same dimensionality as the input.



(a) (b) (c)

Figure 2. Network connection architectures. a) The fully connected architecture. Its weights does not scale well with respect to the number of feature detectors (units). b) The idea of local receptive fields. The number of the weights still grows linearly when the number of feature detectors is increased, but the constant is a much smaller than for the fully connected architecture. c) Combining local detectors with a sliding window search approach. This convolutional approach tries to find a feature from each location of the image. Intuitively, it solves the weight scaling problem by using more computations.

2.2.3 Max Pooling Layer

A max pooling is simply defined as applying a max function for a given pooling window. Pooling windows can be for example 3×3 grid and consecutive windows can be overlapping by defining the step size called the kernel stride. For the backpropagation learning algorithm, introduced in Section 2.3, the error signals are only propagated to the positions of the maximum values [18].

A max pooling layer is used together with a convolutional layer. There are at least two reasons for this. A convolutional layer produces roughly k times overcomplete representation of its input, where k is the number of convolutional filters. On the one hand, a max pooling operation is superior compared to the average subsampling operation for dimensionality reduction of image-like data [18]. On the other hand, the pooling adds a small translational invariance to the network architecture.

If an image is translated only by a few pixels, it is a desired property that the features extracted after the first combination of a convolutional and max pooling layer would be the same. The small translational invariance argument holds also for the representing layers but the max pooling operator is applied in the space of extracted features, which makes it harder to analyze. There are alternatives to the deterministic max pooling. For example, a stochastic pooling, which can be seen as an additional regularization method [19].

2.3 Backpropagation

Backpropagation is a supervised training algorithm for neural networks [1]. The network will calculate a forward propagation for the input at one layer at time so that the activations from the previous layer are used as inputs for the next layer. In each layer, the activations are calculated by multiplying the corresponding weight matrix and adding the biases, and then applying the elementwise activation function. The output layer finally defines the objective function to be minimized.

Backpropagation algorithm states that the correct weight updates for the model parameters can be obtained using a chain rule. While backpropagation sounds at first a very simple and straightforward way to train a model, it has some major drawbacks. For example, vanishing and exploding gradient problems have traditionally prevented neural networks from learning a useful mapping from inputs to outputs. The latter one causes weight updates to be too large, which eventually grow to infinity. Exploding gradients can

be solved by using an activation function that saturates at both ends of its range, for example, a sigmoid function. This, however, leads to the first problem where the gradients of top layers approach zero. The layers below will end up getting an insufficient error signal for updating their weights accordingly, and this problem becomes even worse when the network has many layers.

2.4 Activation Functions

A linear activation corresponds to a weighted sum of its inputs and an offset determined by the biases. A composition of linear functions will also be a linear function, which means that multiple layers of linear functions can always be replaced by just one linear layer. Therefore, a nonlinear activation function is a necessity for deep networks to obtain depth.

2.4.1 Sigmoid

A sigmoid function is a monotonically increasing function, which reaches an asymptote at some finite value as $\pm\infty$ is reached. In the neural network literature, the most commonly used sigmoid function is the standard logistic defined as:

$$\text{logistic}(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})}. \tag{3}$$

As illustrated in Figure 3, the standard logistic curve has the range of $(0, 1)$. This range gives an easy probabilistic interpretation of a unit being active. The function has nearly linear behaviour close to zero and it saturates at both ends.

An important problem occurs, when the weights of the network are initialized by a small random values close to zero. The initial activations of the standard logistic function will then be 0.5 on average. Sigmoids that are symmetric about the origin, e.g. $\tanh(x)$, are preferred because they mitigate the problem of producing always positive outputs and help gradient-based optimization [20].

Another problem on the optimization point of view is that the derivatives for sigmoids, as illustrated in Figure 3, vanish near the saturation points. This makes it harder for a unit to propagate the error signal and move out from the saturation points.

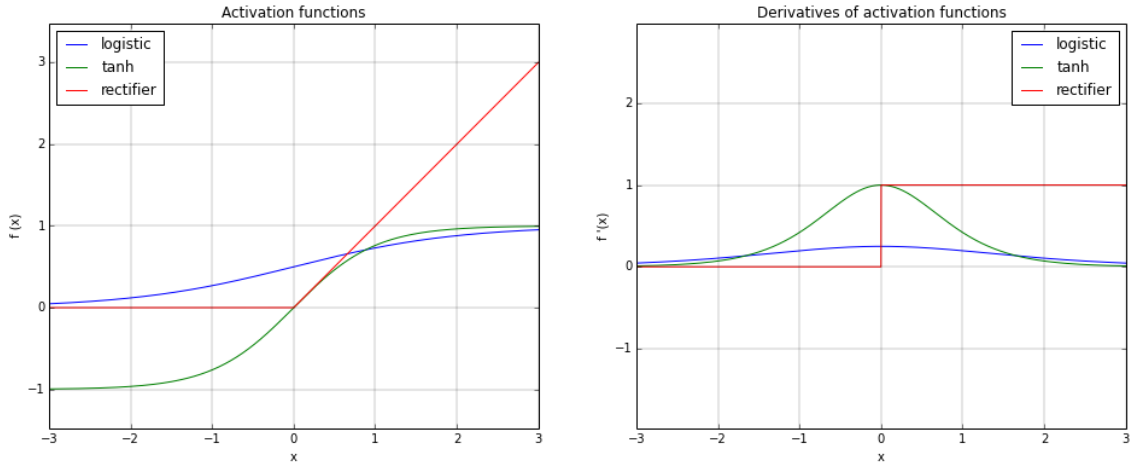


Figure 3. Activation functions (left) and their derivatives (right). Standard logistic and tanh functions are sigmoid functions, which means that they have saturation points at both ends. The rectifier activation has a hard sparsity constraint at zero and it is linear in the positive regime.

2.4.2 Rectifier

A rectifier activation function has been found to improve discriminative performance of convolutional networks [21]. It has also been useful in the context of generative pretraining (Section 3.1) of DNNs [6]. The rectifier nonlinearity is defined as:

$$\text{rectifier}(\boldsymbol{x}) = \max(0, \boldsymbol{x}). \tag{4}$$

It gives true sparsity to the model through the zero constraint, which means that only relatively small set of values are nonzero. Additionally, it does not suffer from the vanishing gradient problem because the function is linear when $x > 0$.

Sparse representation is more biologically plausible and it also helps to disentangle high-dimensional information [22]. Roughly speaking, this means that for a given set of units, most of them are turned off and only a few are activated by some input. One of the reasons why sparsity has interested machine learning researchers is, that sparse features effectively lower the number of dimensions and mitigate the curse of dimensionality. Thus, sparse representation is more robust to small changes in the inputs compared to a dense representation.

It has been suggested that the discontinuity at zero may hurt optimization. However, using a smooth version of the rectifier activation called $\text{softplus}(\boldsymbol{x}) = \log(1 + e^{\boldsymbol{x}})$, which loses the exact sparsity, has performed worse in practise [22].

2.5 Output Layer

2.5.1 Softmax layer for classification

For classification, it is often useful to obtain posterior probabilities for each class rather than just the most probable class. Posterior probabilities have huge practical benefits when, for example, some application-specific tolerance levels are known or when it is important to know solutions that have almost equal likelihood.

For a binary case, the standard logistic activation (Eq. 3) can be used to get the probabilities for both classes. A multiclass case requires a normalization over all the output units, which correspond to the number of different classes. To obtain the probability for a class c, the following softmax function can be used:

$$P(\mathrm{Y} = c|\boldsymbol{x}) = \mathrm{softmax}_c(\boldsymbol{x}) = \frac{e^{\boldsymbol{W}_c\boldsymbol{x}+\boldsymbol{b}_c}}{\sum_j e^{\boldsymbol{W}_j\boldsymbol{x}+\boldsymbol{b}_j}}, \tag{5}$$

where all i output units have corresponding weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$.

To formulate the learning rule for network parameters in each layer, an error function that can be minimized is needed. The natural cost function for softmax, called the Cross-Entropy (CE) [23], is the negative log-likelihood of the right answer. By introducing $\boldsymbol{t}_i$ as the target label for the softmax output $\boldsymbol{y}_i$, the error function corresponds to:

$$E_{CE} = -\sum_{i=1}^{n} \boldsymbol{t}_i \log(\boldsymbol{y}_i), \tag{6}$$

where the sum is calculated over the training set with n samples. At first, it might appear counter-intuitive that only the output unit corresponding to the right answer will contribute changes to the error function. However, the normalization in the softmax units creates nonlocal nonlinearities, which means that changing the output of one unit forces changes in the rest of the units so that the sum of the probabilities equals to one. This makes it perfectly reasonable to minimize the presented error function, which corresponds to the maximum likelihood solution.

2.5.2 Linear layer for regression

Regression problems can use the same underlaying feature extractors, but the output values are real values instead of class labels. This makes it practical to just use a normal linear unit, which corresponds to Eq. 1 without the activation function $g(x)$. The number of linear units in the output layer corresponds to the number of outputs that are required by the task at hand.

For a regression problem, there exists several standard error functions, such as Mean Squared Error (MSE):

$$E_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{t}_i - \boldsymbol{y}_i)^2, \tag{7}$$

where $\boldsymbol{y}_i$ is the output of the linear unit, $\boldsymbol{t}_i$ corresponds to the known true value and the sum is calculated over the training set with n samples.

2.6 Weight Initialization

It has traditionally been difficult to optimize the weights of a network that has multiple nonlinear hidden layers. With large initial weights, the deep network typically finds a poor solution; with small initial weights, the error signal vanishes for lower layers, which makes it infeasible to train a deep network [4].

There have been two ways to deal with this problem. Initially, Hinton et al. suggested that finding initial weights that are close to a good solution would be sufficient for the gradient descent algorithm and they proposed a method to learn initial weights through generative pretraining [3]. Generative pretraining, presented in Section 3.1, introduced a way to utilize unlabeled data, which is a very compelling property. This was clearly a breakthrough in successfully training deep networks and it brought a lot of researchers back to neural networks. The growing attention that deep learning has called since 2006, eventually resulted in finding the rectifier activation function, which now makes it possible to train deep networks even without the pretraining step. When the pretraining can be considered as a trick to help the optimization of deep architectures, the rectifier unit directly solves the optimization issues, such as vanishing gradients and highly entangled representations.

3 REGULARIZATION OF DNN

The expressive power of deep neural networks is two-fold [24]. On the one hand, a huge number of parameters and composition of nonlinearities makes it possible to model very complex function mappings. On the other hand, overcomplete representations, where there are more variables in the model than in the original data, can easily learn trivial solutions like memorizing the input. It is preferred that the network would learn to compute the answer based on extracted features rather than just learn a simple look-up table.

With the expressive power of neural networks, it is easy to reach closer to the zero error rates on a training set by just adding more parameters. However, the network should also be able to generalize well on a new data and not just overfit to the training data. During training, a validation set is used to select the network with the best ability to generalize. To get low error rates also on the validation set, different types of regularization methods are needed for the network. Regularization prevents the network from learning highly entangled features that only work for a specific example. Regularizing the network towards sparse representations assists the learning procedure to employ the full capacity of the network and not only use a few of the units when constructing output values [22].

3.1 Generative Pretraining

Generative pretraining is an unsupervised learning method to obtain initial weights for DNNs. Even though the motivation of pretraining came from an idea of having good enough solution that can be fine-tuned [3], it is important to note that pretraining is also a regularization method. The initial solution is obtained by modeling the input distribution. Then discriminative fine-tuning is based on an assumption that features that can generate the data are also useful to discriminate it. It means that the weights are strongly regularized towards a generative solution and the fine-tuning step can change the weights only slightly to obtain a local minimum in the region.

3.1.1 Restricted Boltzmann Machine

A restricted Boltzmann machine is a particular type of Markov random field that has a two-layer architecture [25]. In the machine the visible, binary stochastic units $\boldsymbol{v} \in \{0,1\}^V$ are connected to hidden binary stochastic units $\boldsymbol{h} \in \{0,1\}^H$, as shown in Figure 4. The

energy of the model with V visible units and H hidden units is defined as follows:

$$\begin{aligned} E(\boldsymbol{v}, \boldsymbol{h}; \theta) = &- \sum_{i=1}^{V} \sum_{j=1}^{H} v_i h_j W_{ij} \\ &- \sum_{i=1}^{V} v_i b_i - \sum_{j=1}^{H} h_j a_j, \end{aligned} \tag{8}$$

where $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{a}\}$ are the model parameters: $\boldsymbol{W}^{V \times H}$ represents the symmetric weights, and $\boldsymbol{b}^V$ and $\boldsymbol{a}^H$ are bias terms.

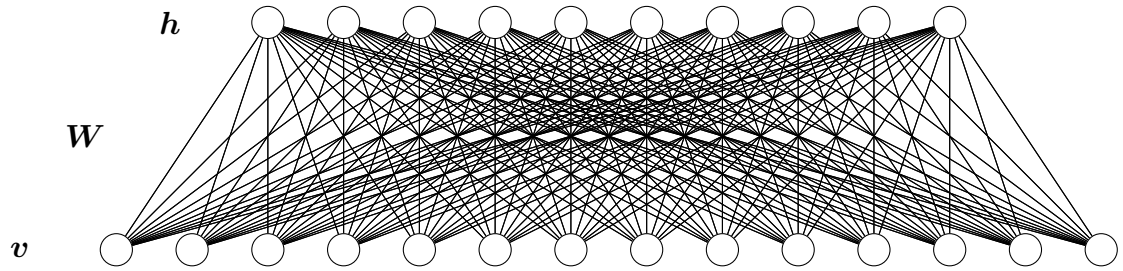


Figure 4. Restricted Boltzmann Machine. The top layer represents a vector of stochastic binary hidden units $\boldsymbol{h}$ and the bottom layer represents a vector of stochastic binary visible units $\boldsymbol{v}$. The symmetric weights $\boldsymbol{W}$ connects these layers.

The conditional distribution for hidden units is as follows:

$$P(h_j = 1|\boldsymbol{v}) = \frac{\exp(\sum_{i=1}^{V} v_i W_{ij} + a_j)}{1 + \exp(\sum_{i=1}^{V} v_i W_{ij} + a_j)} \tag{9}$$

$$= g\left(\sum_{i=1}^{V} v_i W_{ij} + a_j\right), \tag{10}$$

where the activation $g(x) = 1/(1 + \exp(-x))$ is the logistic function. Since $\boldsymbol{v}$ and $\boldsymbol{h}$ play a symmetric role in the energy function, the conditional distribution for visible units is

$$P(v_i = 1|\boldsymbol{h}) = g\left(\sum_{j=1}^{H} h_j W_{ij} + b_i\right). \tag{11}$$

3.1.2 Gaussian RBM

The traditional parametrization of Gaussian RBM is described here. However, as later introduced in Section 3.1.4, a modified version of the energy function is used in the ex-

periments.

To model real-valued continuous data, it is possible to treat the activation values from the Binary RBM as real numbers from the interval $[0, 1]$. However, this approach would be fundamentally flawed because the bias terms of the Binary RBM gives just a linear offset to the energy function. The model is unable to construct an energy surface that is symmetric around the desired value. For example, if a value of 0.7 is desired and corresponds to the lowest energy, then the model should give similar energies for values 0.6 and 0.8.

To address this problem, the hidden units of the RBM remain binary, but the visible units are replaced by linear units with Gaussian noise [4]. The binary RBM's energy function in Eq. 8 is replaced with:

$$E(\boldsymbol{v}, \boldsymbol{h}; \theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H}\frac{v_i}{\sigma_i}h_j W_{ij} + \sum_{i=1}^{V}\frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H}h_j a_j. \tag{12}$$

Here, $\boldsymbol{v} \in \mathbf{R}^V$ denotes the real-valued activities of the visible units. Each visible unit adds a quadratic offset to the energy function, where σ^V controls the width of the parabola. We obtain the following conditional distribution for the visible units:

$$\begin{aligned} P(\boldsymbol{v}|\boldsymbol{h}) &= \frac{\exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{\int_{\hat{\boldsymbol{v}}} \exp\left(-E\left(\hat{\boldsymbol{v}}, \boldsymbol{h}\right)\right) d\hat{\boldsymbol{v}}} \\ &= \prod_{i=1}^{V}\frac{1}{\boldsymbol{\sigma_i}\sqrt{2\pi}}\exp\left(-\frac{1}{2\sigma_i^2}\left(v_i - b_i - \sigma_i\sum_{j=1}^{H}h_j W_{ij}\right)^2\right) \\ &= \prod_{i=1}^{V}\mathcal{N}\left(b_i + \sigma_i\sum_{j=1}^{H}h_j W_{ij}, \sigma_i^2\right). \end{aligned} \tag{13}$$

Similarly to the binary-binary case, this gives the stochastic binary hidden unit where the real-valued visible activity is scaled by the standard deviation

$$P(h_j = 1|\boldsymbol{v}) = g\left(\sum_{i=1}^{V}\frac{v_i}{\sigma_i}W_{ij} + a_j\right). \tag{14}$$

3.1.3 Contrastive Divergence

Because the maximum likelihood learning for energy-based models is often infeasible, an approximation is needed [26]. An efficient learning algorithm, called Contrastive Di-

vergence (CD) [27] can be used. In the binary-binary RBM, this equals to the following learning rule:

$$\Delta \boldsymbol{W} = \alpha \left(E_{P_{data}} \left[\boldsymbol{v}^T \boldsymbol{h} \right] - E_{P_T} \left[\boldsymbol{v}^T \boldsymbol{h} \right] \right) \tag{15}$$

$$\Delta \boldsymbol{a} = \alpha \left(E_{P_{data}} \left[\boldsymbol{h} \right] - E_{P_T} \left[\boldsymbol{h} \right] \right) \tag{16}$$

$$\Delta \boldsymbol{b} = \alpha \left(E_{P_{data}} \left[\boldsymbol{v} \right] - E_{P_T} \left[\boldsymbol{v} \right] \right) \tag{17}$$

where α is the learning rate, $E_P \left[\cdot \right]$ is the expectation with respect to distribution P, P_{data} is the training set empirical distribution and P_T represents a distribution defined by running a Gibbs chain, initialized at the data, for T full steps.

Due to the special bipartite structure of RBM, where there are no connections within a layer, quite an efficient Gibbs sampler exists. Alternating Gibbs sampling, as shown in Figure 5, updates in parallel all the units in one layer given the current states of the units in the other layer, and vice versa (see Eq. 10 and Eq. 11). In CD-T, presented in Algorithm 1, the correlations in the activities of two layers are measured after the first update of the hidden units and again after T steps. The difference of these two correlations provides the learning rule (Eqs. 15, 16, 17) for updating the parameters of the model. Setting $T = \infty$ recovers the maximum likelihood learning and it corresponds to sampling from the model's equilibrium distribution. However, the CD learning with $T = 1$ has been shown to work well [27].

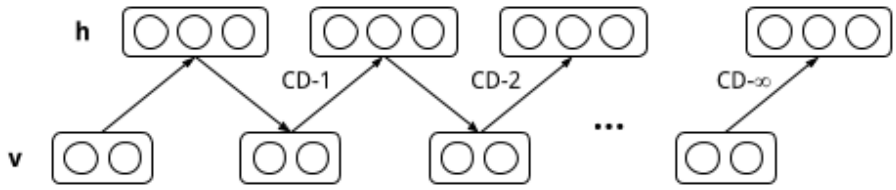


Figure 5. CD-T learning procedure that uses alternating Gibbs sampling. The top layers represent hidden units and the bottom layers visible units. Data is used to initialize the Markov chain.

3.1.4 Denoising Score Matching

With the usual parameterization of the energy function for Gaussian RBM (Eq. 12), it is difficult to learn variances from natural images using Contrastive Divergence. Therefore, those variances are often fixed to unity [28] [29]. To overcome this limitation, we introduce a modified energy function for Gaussian RBM and a Denoising Score Matching (DSM) algorithm.

Algorithm 1: Contrastive Divergence, CD-T

input : $\boldsymbol{v_1}$ – A sample from the training distribution
T – Gibbs sampling steps
α – Learning rate
$\boldsymbol{W}$ – Weight matrix
$\boldsymbol{a}$ – Bias vector for hidden units
$\boldsymbol{b}$ – Bias vector for visible units

output: $\boldsymbol{W}$ – Updated weight matrix
$\boldsymbol{a}$ – Updated bias vector for hidden units
$\boldsymbol{b}$ – Updated bias vector for visible units

compute expectations $\boldsymbol{p_{h_1}} \leftarrow P(\boldsymbol{h} = 1|\boldsymbol{v_1})$ for all hidden units

for $i = 1$ **to** T **do**
  sample $\boldsymbol{h_i}$ from $\boldsymbol{p_{h_i}}$
  compute expectations $\boldsymbol{p_{v_{i+1}}} \leftarrow P(\boldsymbol{v} = 1|\boldsymbol{h_i})$ for all visible units

  sample $\boldsymbol{v_{i+1}}$ from $\boldsymbol{p_{v_{i+1}}}$
  compute expectations $\boldsymbol{p_{h_{i+1}}} \leftarrow P(\boldsymbol{h} = 1|\boldsymbol{v_{i+1}})$ for all hidden units
end

$\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha(\boldsymbol{v_1}^T \boldsymbol{p_{h_1}} - \boldsymbol{p_{v_{T+1}}}^T \boldsymbol{p_{h_{T+1}}})$
$\boldsymbol{a} \leftarrow \boldsymbol{a} + \alpha(\boldsymbol{p_{h_1}} - \boldsymbol{p_{h_{T+1}}})$
$\boldsymbol{b} \leftarrow \boldsymbol{b} + \alpha(\boldsymbol{v_1} - \boldsymbol{p_{v_{T+1}}})$

Score matching (SM) is an alternative to CD. Energy-based models, whose partition function is intractable, can be estimated by minimizing the expected squared distance between the gradient of the log-density given by the model and the gradient of the log-density of the observed data [30]. Denoising Autoencoder (DAE) is a deep learning model that forces the hidden layer to discover more robust features by reconstructing the input from a corrupted version of it [31]. DAEs have proven to be an empirically successful alternative to RBMs for pretraining deep networks. Pascal Vincent showed that there is equivalence between DAE and Gaussian RBM trained using DSM [32] which leads to the following objective function:

$$J(\theta) = E_{q_\sigma(\tilde{\boldsymbol{v}},\boldsymbol{v})}\left[\frac{1}{2}\left|\left|\psi(\tilde{\boldsymbol{v}};\theta) - \frac{\partial \log q_\sigma(\tilde{\boldsymbol{v}}|\boldsymbol{v})}{\partial \tilde{\boldsymbol{v}}}\right|\right|^2\right], \tag{18}$$

where $E_{q_\sigma(\tilde{\boldsymbol{v}},\boldsymbol{v})}[\cdot]$ is the expectation with respect to the Gaussian noise model. The visible units are corrupted using additive Gaussian noise: $\tilde{\boldsymbol{v}} = \boldsymbol{v} + \epsilon, \epsilon \sim \mathcal{N}(0, \boldsymbol{\sigma}^2\boldsymbol{I})$. Thus, the latter term, the gradient of the Parzen window density estimator with respect to the corrupted visible units, corresponds to

$$\frac{\partial \log q_\sigma(\tilde{\boldsymbol{v}}|\boldsymbol{v})}{\partial \tilde{\boldsymbol{v}}} = \frac{1}{\sigma^2}(\boldsymbol{v} - \tilde{\boldsymbol{v}}). \tag{19}$$

The gradient of the log density with respect to the corrupted visible units $\tilde{\boldsymbol{v}}$, is called score: $\psi(\tilde{\boldsymbol{v}};\theta) = \frac{\partial \log p(\tilde{\boldsymbol{v}};\theta)}{\partial \tilde{\boldsymbol{v}}}$. Now we introduce the following energy function of Gaussian RBM to achieve the equivalence with DAE: [1]

$$E(\boldsymbol{v},\boldsymbol{h};\theta) = -\frac{1}{\boldsymbol{\sigma}^2}(\boldsymbol{v}^T\boldsymbol{b} + \boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h} + \boldsymbol{h}^T\boldsymbol{a} - \frac{1}{2}\boldsymbol{v}^T\boldsymbol{v}) \tag{20}$$

$$P(\boldsymbol{h}|\boldsymbol{v}) = g\left(\frac{\boldsymbol{W}\boldsymbol{v} + \boldsymbol{a}}{\boldsymbol{\sigma}^2}\right) \tag{21}$$

$$P(\boldsymbol{v}|\boldsymbol{h}) = \mathcal{N}\left(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \boldsymbol{\sigma}^2\right) \tag{22}$$

$$\psi(\boldsymbol{v}) = -\frac{1}{\boldsymbol{\sigma}^2}(\boldsymbol{v} - \boldsymbol{b} - g\left(\frac{\boldsymbol{W}\boldsymbol{v} + \boldsymbol{a}}{\boldsymbol{\sigma}^2}\right)\boldsymbol{W}^T), \tag{23}$$

where $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{a}, \boldsymbol{\sigma}\}$ are the model parameters similar to binary RBM with an added variance term $\boldsymbol{\sigma}^V$ for the visible units. Substituting Eq. 19 and Eq. 23 in Eq. 18, leads to the final objective function, which can then be minimized by using the gradient descent algorithm.

[1] This is not the same energy function as presented in Eq. 4.3. in [32] because in that energy function there is no hidden variables and, therefore, is not an RBM. Parameterization in Eq. 20 is proposed by Ian Goodfellow in Pylearn2 [33] implementation of Gaussian RBM.

3.1.5 Gibbs Sampling for Convolution and ReLU Units

As with the standard fully-connected RBM in Section 3.1.1, we can also perform block Gibbs sampling with the convolutional architecture. The conditional distributions in Eqs. 10 and 11 will have the following changes:

$$P(h_{ij}^k = 1|v) = g\left((\boldsymbol{W}^k * v)_{ij} + a_{ij}^k\right) \tag{24}$$

$$P(v_{ij} = 1|h) = g\left(\left(\sum_k \tilde{\boldsymbol{W}}^k * h^k\right)_{ij} + b_{ij}\right), \tag{25}$$

where the matrix multiplication is replaced with the convolutional operation. The other direction is obtained by flipping the weights horizontally and vertically (denoted as $\tilde{\boldsymbol{W}}^k$).

The probabilistic interpretation is valid for binary units and Bernoulli distribution can be used to sample values. For rectified activation, similar kind of conditional distributions cannot be formed because $\max(x, 0)$ is not in the exponential family. However, a noisy version of rectified activation, $\max(0, \boldsymbol{x} + \mathcal{N}(0, \sigma(\boldsymbol{x})))$ where $\mathcal{N}(0, V)$ is Gaussian noise with zero mean and variance V, can be used to do the required sampling for CD algorithm. In several tasks, it has been showed to work better than the binary units [6].

3.1.6 Deep Belief Nets

An efficient way to learn a complicated model is to combine a set of simpler models that are learned sequentially. In 2006 Hinton et al. introduced Deep Belief Nets (DBN) [3], with a learning algorithm that greedily trains one layer at a time, exploiting the unsupervised learning for each layer. Each layer in a DBN captures high-order correlations between the activities of hidden features in the layer below. A key feature of this algorithm is its greedy layer-by-layer training that can be repeated several times. Variational lower-bound justifies the greedy layerwise training of RBMs [3].

The main building block of the DBN is a Restricted Boltzmann Machine. The idea of greedy learning algorithm is quite simple. Train the first RBM normally using the data in the visible layer. Then freeze the learned parameter vector and use the hidden layer activations as input data when training the second RBM. This process is outlined in Figure 6. Finally, a normal feed-forward neural network can be initialized using the weights learned by the pretrained RBMs.

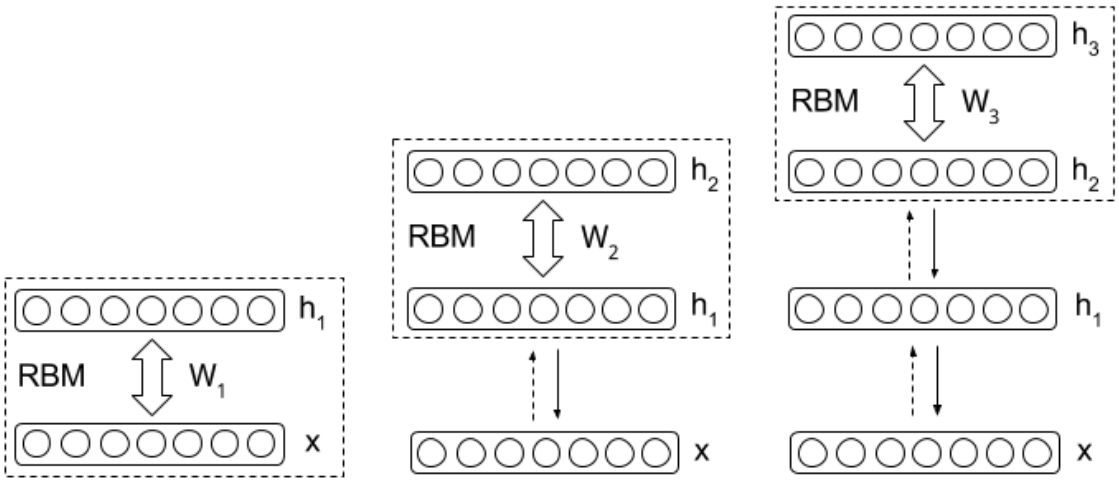


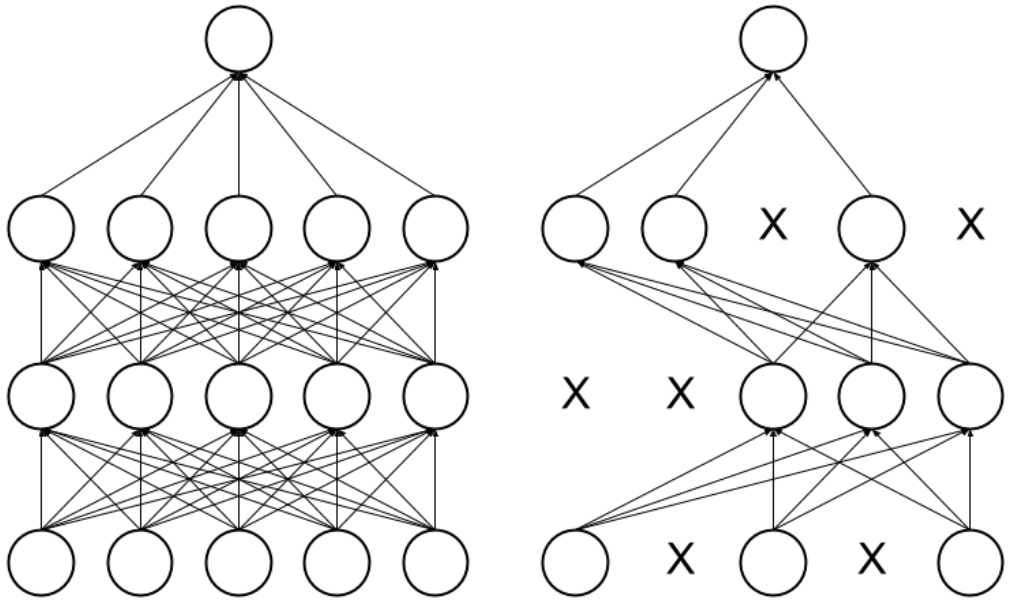
Figure 6. Learning with Deep Belief Nets. The first Restricted Boltzmann Machine is trained to model the raw input $\boldsymbol{x}$ as its visible layer. The second RBM is trained using the transformed data $\boldsymbol{h_1}$ from the previous layer as training examples for its visible layer. This process can be repeated to increase the depth of the DBN. $\boldsymbol{h_i}$ corresponds to learned features and $\boldsymbol{W_i}$ to the learned weight matrix by the i-th RBM. Finally, a discriminative fine-tuning can be performed by adding a final layer that represents the desired outputs and backpropagating the error derivatives.

3.2 Dropout

Dropout is a technique that prevents overfitting and provides an efficient way of doing model averaging. The first phase, illustrated in Figure 7b, corresponds to dropping out units during training. Each unit in a layer, with all its incoming and outgoing connections, is dropped out with a fixed probability p independent of the other units. With $p = 0.5$, a neural net with n units can be seen as a collection of 2^n possible thinned networks that all share weights. At test time, a very simple approximate model averaging method is performed. The full network without dropout is used and the weights are multiplied by p. This ensures that the expected output of any hidden unit is the same as the actual output at test time [34].

3.3 Weight-Decay

Traditionally a small amount of L2 weight-decay has been used to prevent weights from growing arbitrarily large. When trying to optimize a high-dimensional parameter space, the curse of dimensionality is a serious problem. The gradient descent approach may easily find the closest local minima by just growing a few of the weights. This phenomenon is not desired because only those few connections would overtake all the others and the



(a) (b)

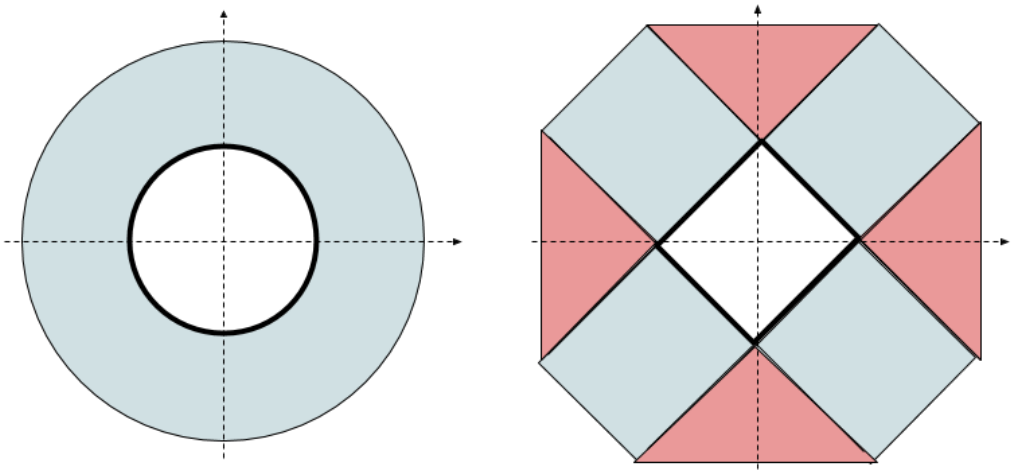
Figure 7. Dropout applied to a neural network. a) The standard fully connected neural network. b) Dropout is applied and the resulting network is a thinned version of the original network, which is used during the training. At the test time, the fully connected network on the left is used with weights multiplied by the probability of dropping a unit.

expressive power of a neural network by having large amount of parameters would be violated. [35]

An easy way to increase sparsity in a model is to use the L1 weight-decay [36, 37]. By penalizing weights with the L1-norm, the weight-decay induces sparsity to the model, as illustrated in Figure 8b, in the sense that it forces many of the weights to become exactly zero, while allowing a few of the weights to grow large. Circle-shaped L2-norm, projects all values perpendicular with respect to the norm edge. The square form of the L1-norm, however, forms also areas where values are projected to single points, to the corners of the L1-norm. Those corner points correspond to locations, where one of the coordinates is zero. This is the reason, why using L1-norm in a weight-decay induces sparsity to the model.

3.4 Max Kernel Norm

When using L2 weight-decay, a single large weight in a layer can have too much effect for the whole output of that layer and the learning will eventually just modify that weight [37].



(a) (b)

Figure 8. Two dimensional L2 and L1 unit norms and projections. a) Circle-shaped L2-norm. b) L1-norm, is a square rotated by 45 degrees. Gray areas illustrate regions where values are projected perpendicular with respect to the norm edges. Red areas on the right, however, represent areas where all values are projected to single points, to the corners of the L1-norm. Those corner points correspond to locations, where one of the coordinates is zero.

By setting a maximum kernel norm value, a small constant is reduced from the weight that achieves the maximum value [38]. This approach does not penalize all weights, like the traditional weight-decay, if only a few of the weights keeps growing larger. In the minimization perspective, the weight decay prefers a minimum with a smaller set of weights. Max kernel norm does not make this kind of choice between the minima given that the weights are under a given threshold.

3.5 Data Augmentation

A very fundamental approach to improve generalization is getting more training data. If a training set is scarce and has some transformation-invariance properties, generating additional data may be useful [39, 40]. We randomly added small variations to the training set, corresponding to rotation and scale. The amount of variation for each distortion were uniformly sampled from the interval, which was determined using the validation set. Small distortions provided additional training data that is consistent with the original data distribution. For images, even small random rotation could be enough to prevent deep network to memorize its input.

3.6 Optimization

3.6.1 Stochastic Gradient Descent

The standard Gradient Descent (GD) is a first-order optimization algorithm. The algorithm uses iterative steps to find a local minimum of a function. It utilizes the gradient of the function to obtain the direction where the function decreases fastest. Excluding a few special cases, this direction rarely directly points to the minimum when the minimization process starts. However, using the iterative scheme, the algorithm is guaranteed to find a local minimum for convex functions when sufficiently small step size is used [41].

Choosing the step size, also known as the learning rate in the machine learning community, is difficult because the function to be minimized in deep learning is rarely convex [1]. For example, optimizing from an initial starting point to a local minimum, there can be a long flat ravine which suddenly changes to a steep slope. These contours require very different magnitudes of learning rates to get both a correct solution and acceptable optimization time. This problem is generally hard when information on second-order derivative is not available, like in the case of using backpropagation. Using exponential decay or scheduled learning rate and momentum optimization, introduced in Section 3.6.2, are ways to mitigate this problem [42].

The GD update rule for minimizing a function f to estimate parameters θ is defined as follows:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} f(\boldsymbol{x}; \theta_t), \tag{26}$$

where $\alpha > 0$ is the learning rate, t is the iteration step, $\nabla_\theta f$ is the gradient of f with respect to θ and $\boldsymbol{x}$ is the training set. One pass through the training set is called an epoch. In the gradient descent, an epoch corresponds to one update of the parameters. An immediate problem occurs, when the training set grows large: It is infeasible to compute the gradient against a huge amount of data. By dividing the training set into L mini-batches $\boldsymbol{x}^i$, and estimating the gradient by summing these mini-batches, the batch GD is obtained:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{L} \sum_{i=1}^{L} \nabla_{\theta_t} f(\boldsymbol{x}^i; \theta_t). \tag{27}$$

While the properties of this optimization algorithm are well known [43], the average of the gradients over the entire training set still involves a burdening computation.

The batch version of Stochastic Gradient Descent (SGD), presented in Algorithm 2, is a

slight modification of batch GD where the average in Eq. 27 is replaced by L separate parameter updates in one epoch. This simplification relies on the hope that the random noise introduced by the modification will not change the average behaviour of the algorithm [43, 44]. In the literature, there exists a huge amount of empirical evidence on the success of SGD in large-scale machine learning problems.

Algorithm 2: Stochastic Gradient Descent (batch)

input : $\boldsymbol{x}$ – Training set
θ – Initial parameters
α – Learning rate
$\nabla_\theta f$ – Gradient of function f to be minimized with respect to parameters
output: θ – Estimated parameters

repeat
  generate a set of L randomized mini-batches from $\boldsymbol{x}$
  for $i = 1$ **to** L **do**
    take mini-batch $\boldsymbol{x}^i$ from the set of randomized mini-batches
    calculate gradient $\leftarrow \nabla_\theta f(\boldsymbol{x}^i; \theta)$
    calculate $\theta \leftarrow \theta - \alpha *$ gradient
  end
until *Stopping condition is met*

3.6.2 Momentum

Momentum-based acceleration methods, like Nesterov's Accelerated Gradient (NAG), have become more popular on training deep networks with SGD. NAG is capable of accelerating directions of low-curvature, which is particularly useful on training deep networks. It explores new regions of the parameter space that are higher-quality local minima [45]. The NAG update rule is given by:

$$v_{t+1} = \mu v_t - \alpha \nabla_{\theta_t} f(\boldsymbol{x}; \theta_t + \mu v_t) \tag{28}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{29}$$

where $\mu \in [0, 1]$ is the momentum coefficient. Intuitively, this tells us first to take a step in the direction of the velocity v_t, which is the direction of the previous parameter update, and then calculate the gradient update. By setting $\mu = 0$, this becomes the normal gradient descent update rule.

4 EXPERIMENTS AND RESULTS

In the experiments, the following questions are tested on a convolutional architecture:

1. The effect of pretraining.
2. Dropout during pretraining.
3. Common tricks used in practical applications of purely supervised learning.

To test the effect of pretraining, the weights of ReLU Convolutional Deep Belief Nets (CDBN) are first initialized using unsupervised learning. Then the weights of CDBN are fine-tuned and its classification results are compared to the ReLU CNN, which does not contain the pretraining step.

In the second phase, we tested dropout during the pretraining step. We wanted to find out how this effective regularization method, normally used in supervised learning, behaves in generative pretraining.

We also compared pretrained models with a purely supervised model. In the first two tests on MNIST dataset, models were fine-tuned and trained without additional regularizers. However, in the third test, we wanted to find out whether the pretraining is still helpful if many of the popular regularization methods and optimization techniques are present, such as, dropout, data augmentation, weight decay and momentum optimization.

The amount of labeled data in the supervised learning step is varied in each test. The motivation for this is to resemble real use cases where there are more unlabeled than labeled data available. Also, to find out whether there is an amount of labeled data that is enough to damp the effect of pretraining. Based on these results, we compared the best pretrained model and the best baseline on a natural image classification task.

4.1 Models

4.1.1 Baseline: ReLU CNN

Our baseline model consist of convolutional neural network with rectified linear activation. CNN model consist of a convolutional layer followed by a max pooling layer, which

is particularly useful for image data because it introduces a small amount of translational invariance.

We selected rectified linear units because they have been show to be superior compared to sigmoid units [21, 6] and we wanted to compare our method to an up-to-date baseline. Another important point is that the ReLU unit has also solved the problem of vanishing gradient, thus making the pretraining step less critical for training deep models. By picking a very strong baseline, we are interested to find out whether and in which conditions the pretraining would still be useful in practical cases.

4.1.2 Proposed: ReLU CDBN

To be able to pretrain rectified linear units, we needed to modify the original CDBN model [14]. The probabilistic max pooling of CDBN is formulated so, that there is no straightforward way to change the sigmoid activation function. For that reasons, the ReLU version of CDBN uses the traditional deterministic max pooling.

Other than different pooling and activation functions, the training of ReLU CDBN follows greedy layer-wise pretraining: The first layer Gaussian-ReLU RBM is trained using DSM and the second layer is trained with CD-1, with rectified linear visible and hidden layers. Once the weights are obtained from pretraining, we fine-tune the model with the same supervised learning step that is used in the baseline.

4.2 Pretraining on Handwritten Digits

4.2.1 MNIST Dataset

The MNIST handwritten digit classification task [46], which is a widely-used benchmark in deep learning, is used to evaluate performances of the models. There is a total of 70,000 grayscale images, which is traditionally splitted into 50,000 training, 10,000 validation and 10,000 test sets. In a sense, the dataset is relatively simple having only ten classes and the data consist of pen strokes on a constant black background. However, the amount of data it contains classifies it into the group of a large scale machine learning task where deep learning excels. A nice property of the dataset is that it is so popular that virtually every new idea or method in deep learning is first tested on MNIST. We do not use

any preprocessing steps for this dataset and the images are fed into the models directly by using the grayscale intensities from the interval 0 to 1. Examples of the dataset are illustrated in Figure 9.

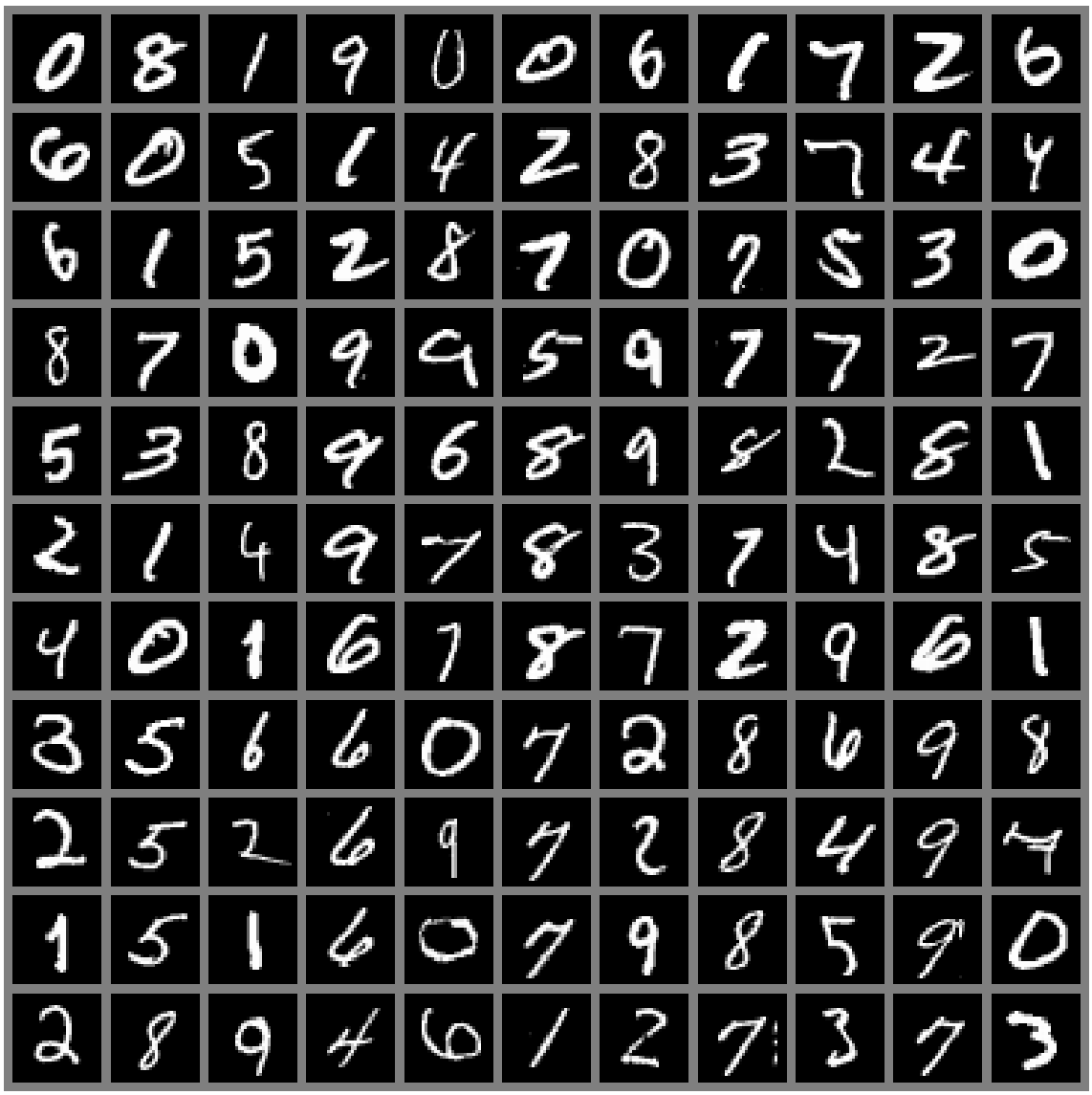


Figure 9. Examples of handwritten digits in MNIST dataset.

4.2.2 Network Architecture and Setup of Learning

The architecture used in pretraining experiments with MNIST data consists of two convolutional hidden layers and a softmax classification layer. The first and the second layer have both 48 feature maps, which are formed using 9×9 and 7×7 convolutional kernels

respectively. After the convolutions, the both subsampling layers do max pooling in non-overlapping regions of 2×2, The convolutional operation in the second layer adds full zero-padding for the input images to handle the boundary pixels. The described network architecture is illustrated in Figure 10.

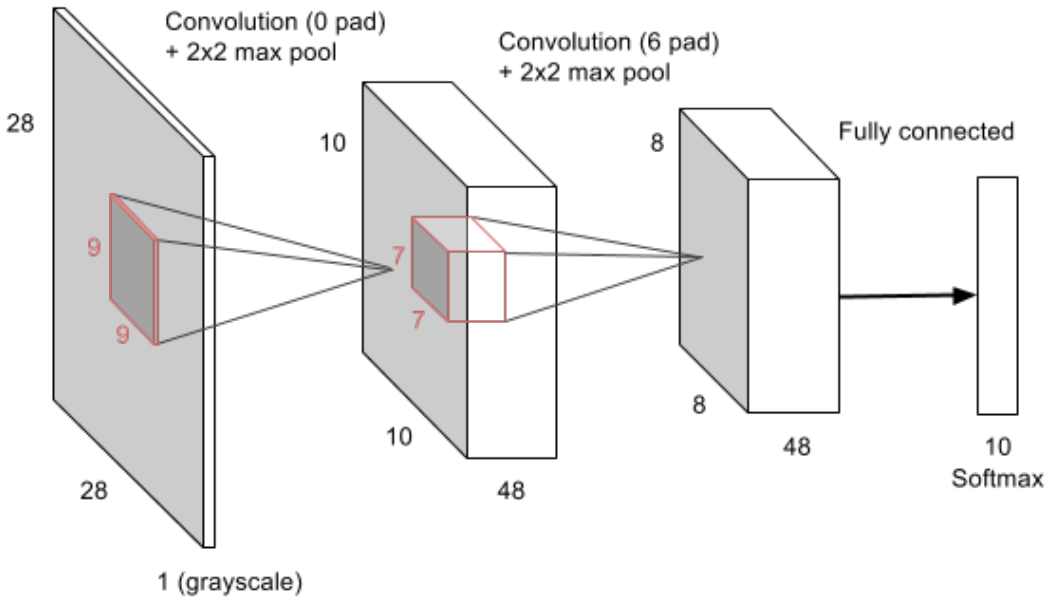


Figure 10. Convolutional Neural Network architecture in pretraining experiments with MNIST data. Hidden layers correspond to convolutional layers followed by 2×2 max pooling. The softmax is used as the output layer.

The first layer Gaussian-ReLU RBM was trained using Denoising Score Matching with a 0.4 Gaussian corruptor. We experimented with different Gaussian corruptor values and found out that values close to 0.4 gave similar performance with very little difference, but after a bigger threshold, values like 0.8 and 0.1, the performance started to decrease. The second layer ReLU-ReLU RBM was trained on the activations of the Gaussian-ReLU RBM using CD-1 algorithm. Convolutional weights for the both layers were initialized using the uniform distribution from the interval $[-0.01, 0.01]$. The hidden and visible biases were set to 0.

The random weights in supervised training are drawn from the uniform distribution from the interval $[-0.05, 0.05]$. While training with smaller weights converged into a meaningful solution in unsupervised learning, we found out in our initial experiments that slightly larger weights were necessary for supervised learning. The bias values were set to zero.

In the supervised training, these first two tests used the plain batch version of stochastic gradient descent algorithm. The batch size for SGD was set to 128 images to speed up the computations. The learning rate was set to 0.05, which was selected using a grid search, and all models were trained for 300 epochs. Each model in the comparison was selected based on the classification accuracy on the validation set.

Dropout, data augmentation and Nesterov's momentum optimization were added in the regularization test. Initially, the momentum was set to 0.5, and during the first 150 epochs, it exponentially grew to its maximum value of 0.9. Dropout was set to 0.8 for the both layers. To implement data augmentation, scale and rotation were sampled from the uniform distributions of $[0.9, 1.1]$ and $[-8°, 8°]$ respectively, which were determined by the validation set. We also experimented with momentum values 0.7, 0.95 and 0.99, dropout value 0.5 and translation on data augmentation, but none of these improved the performance.

4.2.3 Pretraining Results

In the first experiment, we tested the effect of pretraining with and without dropout. For the supervised step, we did not utilize any regularization methods. We varied the size of the training set to see whether the effect of pretraining fades when more labeled data are available. Splits of 1,000 and 5,000 examples were randomly sampled, and we report the average error and standard deviation over the splits. For the full dataset, the best result is reported.

Results are summarized in Table 1. The experiments showed that without additional regularization, pretrained models gave generally lower average error rates on handwritten digits classification task. A bit surprisingly, the relative difference between pretrained and randomly initialized models are the largest for models trained with the full training set. The lowest error of 0.48%, obtained by ReLU-CDBN with dropout pretraining, is comparable to the state-of-the-art for this dataset. The training set size of 5,000 followed the same pattern, where the both pretrained models got lower average errors. In contrast, for the smallest training set, the model with dropout pretraining digressed from the trend resulting worse error rate than the baseline.

The learned first layer convolutional filters are illustrated in Figure 11. Initial filters of CDBNs correspond to sharp pen strokes and point filters that include almost zero amount of noise. With 1,000 images, filters of the baseline contains still a lot of noise. The noise level of ReLU CNN filters is reduced when the model is trained with more examples. For

pretrained models, the progression is different: Sharp filters become more fuzzy and point filters evolve to resemble Gabor-like filters.

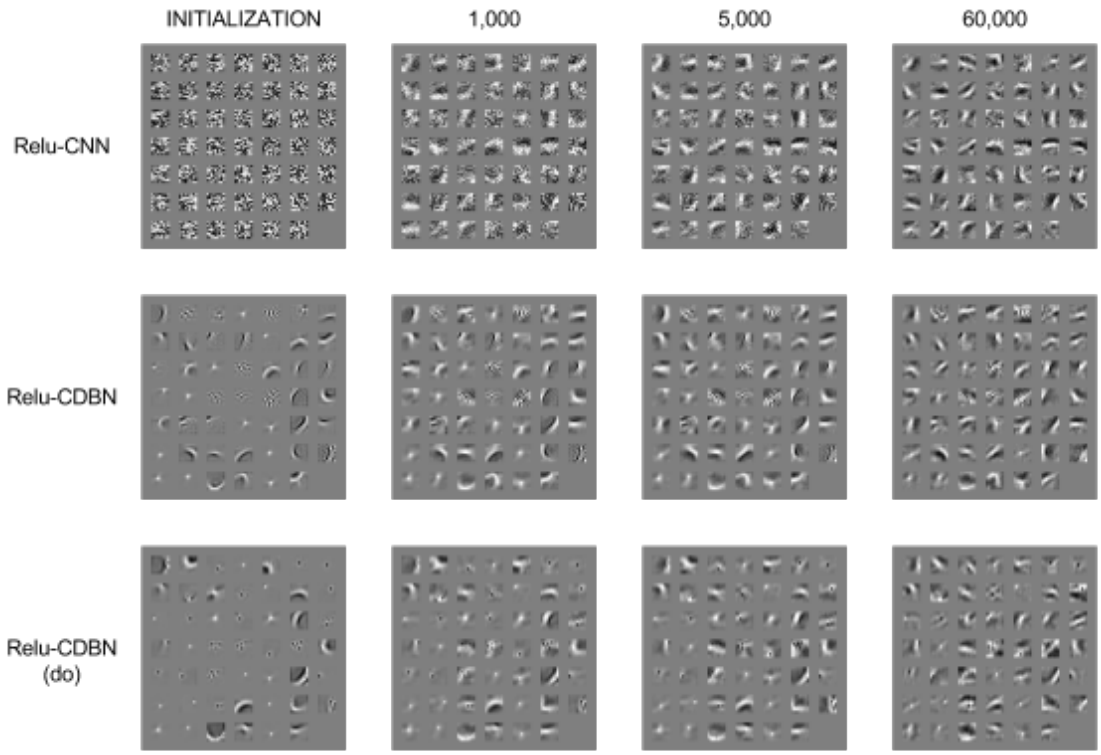


Figure 11. Learning progress of the first layer weights for models on MNIST dataset. Rows correspond to models and the learning progress with respect to training set size is visualized in each column. Dropout during pretraining is denoted as “(do)".

4.2.4 Regularization Results

In Table 2 the regularization results are summarized. The effect of regularization was also tested with the same three models than in the last section. Overall, the biggest improve-

Table 1. Pretraining test errors for models on MNIST dataset. For sampled folds, the average error and the standard deviation are reported.

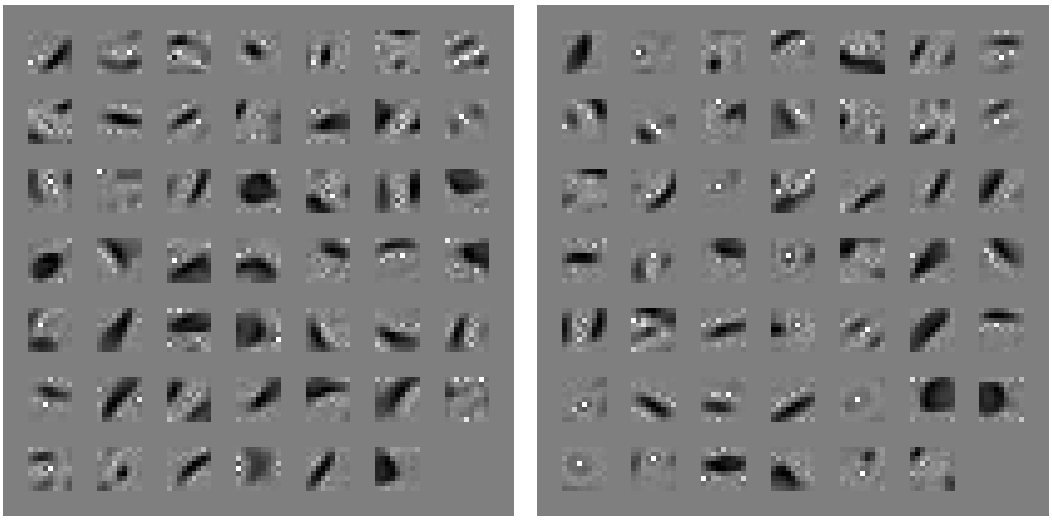
Labeled training samples	1,000	5,000	60,000
ReLU-CNN	4.92% (0.35)	1.98% (0.08)	0.61%
ReLU-CDBN	4.53% (0.30)	1.85% (0.14)	0.51%
ReLU-CDBN (do)	4.94% (0.44)	1.88% (0.12)	0.48%

ment was with the fully supervised CNN model and it was the best model for the both smaller size training sets. The difference in average error for ReLU-CNN got approximately two percent lower by using just standard regularization and optimization methods. Using the full training set, the error increased for the pretrained models, and the model trained with dropout pretraining from the last section remained the best for this dataset.

Table 2. High regularization and momentum optimization test error for MNIST dataset. For sampled folds, the average error and standard deviation are reported.

Labeled training samples	1,000	5,000	60,000
ReLU-CNN	2.87% (0.32)	1.31% (0.07)	0.56%
ReLU-CDBN	3.21% (0.37)	1.35% (0.04)	0.65%
ReLU-CDBN (do)	3.44% (0.60)	1.39% (0.09)	0.63%

The first layer filters trained with the full training set differ largely compared to ones from the previous test. Trained with the full training set, the randomly initialized CNN weights and pretrained CDBN weights are visually very similar. These mostly dark filters with single white point are illustrated in Figure 12. Misclassified examples by our best model for MNIST dataset are shown in Figure 13. The CDBN model with dropout pretraining got only 48 errors on 10,000 images. By using posterior probabilities, we calculated that 40 of these errors were predicted correctly by the network's second guess.



(a) CNN weights. (b) CDBN weights.

Figure 12. First layer weights of regularized CNN and CDBN learned on MNIST (60,000 examples).

Figure 13. Misclassified examples on MNIST by our best model. Green and red colored numbers corresponds to true and predicted classes. Examples marked by a yellow bounding box were also misclassified by the network's second guess (yellow number).

We conclude the experiments on MNIST by comparing our models to the state of the art. The results in Table 3 shows that our models are comparable to the best performing models for this dataset.

Table 3. Comparing the best models to the state-of-the-art on MNIST dataset.

Model	Error
ReLU-CNN	0.56%
ReLU-CDBN	0.51%
ReLU-CDBN (do)	0.48%
DNN + Dropconnect [47]	0.21%
Multi-column DNN [48]	0.23%
Convolutional Kernel Networks [49]	0.39%
Maxout Networks [8]	0.45%
Sigmoid-CDBN + SVM [14]	0.82%

4.3 Natural Image Classification

4.3.1 STL-10 Dataset

While MNIST dataset is great for model development the real interest is to recognize objects from natural images, for example pictures taken by mobile phones. There is a huge amount of variation in natural data which makes it much harder problem for computer vision. STL-10 dataset [50, 51] was selected to experiment models with real-world images.

STL-10 dataset contains very few labeled training examples. The training set is organized into 10 predefined folds. Each fold consist of 1,000 images of size 96×96 categorized into 10 classes. The dataset has totally 5,000 labeled training images, which means that the folds are partially overlapping with each other. The evaluation protocol is defined as follows:

1. Perform unsupervised training on the unlabeled data.
2. Perform supervised fine-tuning separately for each predefined fold.
3. Calculate accuracy for each fold on the full test set. Report average and standard deviation of the results.
4. For each training set fold, the remaining 4,000 images are used as the validation set. This convention is reported by the state-of-the-art solution for STL-10 dataset. [52]

The unlabeled set and the test set contain 100,000 and 8,000 images respectively. What is particularly interesting in this dataset is the 1-to-8 ratio of training vs. test images. The large set of unlabeled images holds potentially a huge amount of information that is required to perform well on this dataset. Examples of the dataset grouped by class labels are illustrated in Figure 14.

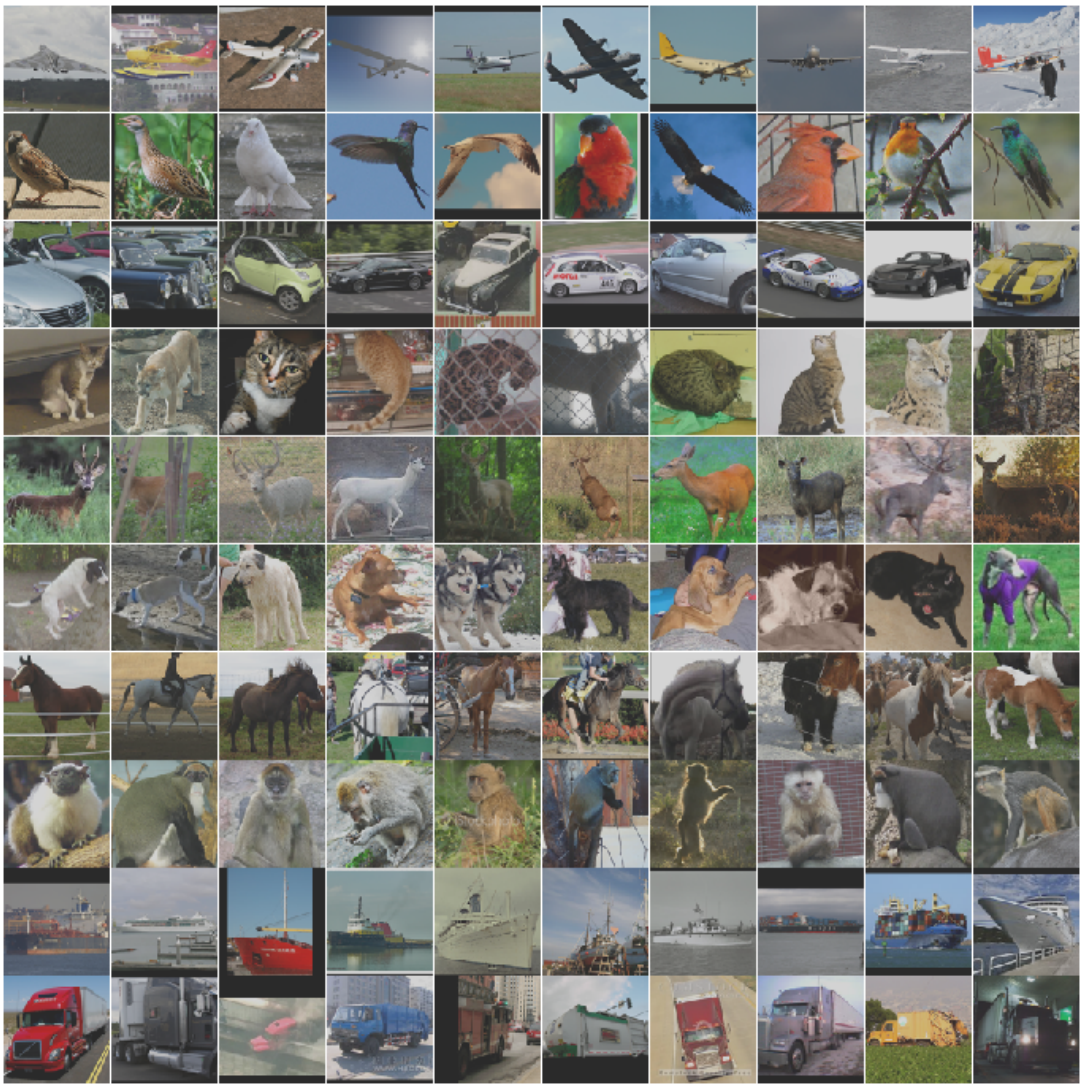
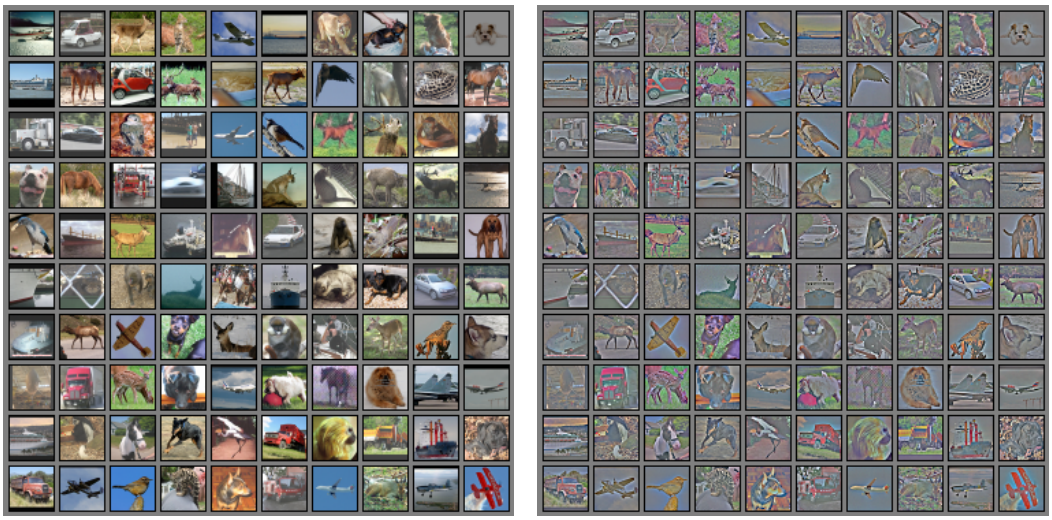


Figure 14. Examples of STL-10 dataset. Rows correspond to images that belong to the same class. Classes from top to down are airplane, bird, car, cat, deer, dog, horse, monkey, ship and truck. [51]

Input images of 32×32 size, are obtained by first downsampling the original images by a factor of 3. This step significantly speeds up the computations. Then Zero-phase Component Analysis (ZCA) whitening [53] is applied to reduce global pixel correlations while changing the data as little as possible in order to maintain local correlations. Convolutional layers of the network extract local features from images, so it is very important

property that the ZCA-whitened images, illustrated in Figure 15b, still resemble normal images.



(a) Downsampled examples. (b) ZCA-whitened examples.

Figure 15. Preprocessing steps for STL-10 dataset. Downsampled 32×32 examples are on the left and examples after ZCA whitening are on the right. Whitened images are the inputs of the neural network.

4.3.2 Network Architecture and Details of Learning

The architecture used in STL10 experiments consists of two convolutional hidden layers, two fully connected rectified linear layers and a softmax classification layer. The first and the second convolutional layer have both 96 feature maps, which are formed using 9×9 and 7×7 convolutional kernels respectively. 2×2 non-overlapping max pooling is performed after the convolutional layers similar to MNIST architecture. For the both convolutional layers, a full zero-padding for input images was added to handle the boundary pixels. The described network architecture is illustrated in Figure 16.

The first layer Gaussian-ReLU RBM was trained using Denoising Score Matching with a 0.4 Gaussian corruptor. The second layer ReLU-ReLU RBM was trained using CD-1 algorithm. For both RBMs, the convolutional weights were initialized using the uniform distribution from the interval $[-0.01, 0.01]$. The fully connected layers were not pretrained. The random weights in supervised training were drawn from the uniform distribution from the interval $[-0.05, 0.05]$. Hidden and visible biases were initialized to zero for convolutional layers and 0.05 for the fully connected layers. By setting a non-zero bias for fully connected ReLU layer helped to propagate the error signal in the deep

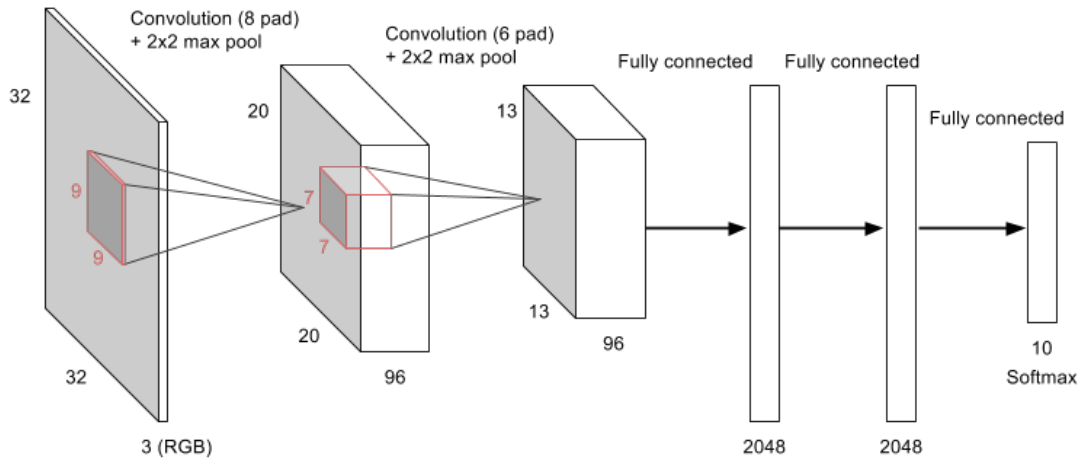


Figure 16. Convolutional Neural Network architecture in STL10 experiments. First two layers correspond to convolutional layers followed by 2×2 max pooling. Last two hidden layers are fully connected layers with ReLU activation and the softmax is used as the output layer.

architecture.

In the supervised training, the batch size for SGD was set to 100. Learning rate was set to the relatively small value of 0.005, and models for each fold were trained for 3000 epochs. The network was much larger than in the MNIST experiments. Therefore, a small learning rate and large number of epochs provided an optimization strategy that relatively slowly but steadily approached to a local minimum. This required more training time in a single run, but avoided costly hyperparameter search. Each model in the comparison was selected based on the classification accuracy on the validation set.

Dropout, data augmentation and Nesterov's momentum optimization were used. Initially the momentum was set to 0.5, and during the first 500 epochs, it exponentially grew to its maximum value of 0.9. Dropout was set to 0.8 for the convolutional layers and 0.5 for fully connected layers. For each training image, scale and rotation were sampled from the uniform distributions of $[0.9, 1.1]$ and $[-8°, 8°]$ respectively.

4.3.3 Results

We follow the evaluation protocol for STL-10, introduced in Section 4.3.1, and report the average classification accuracy and standard deviation over the predefined folds. The proposed method got 1.64% better results than the baseline. While the result for CDBN is

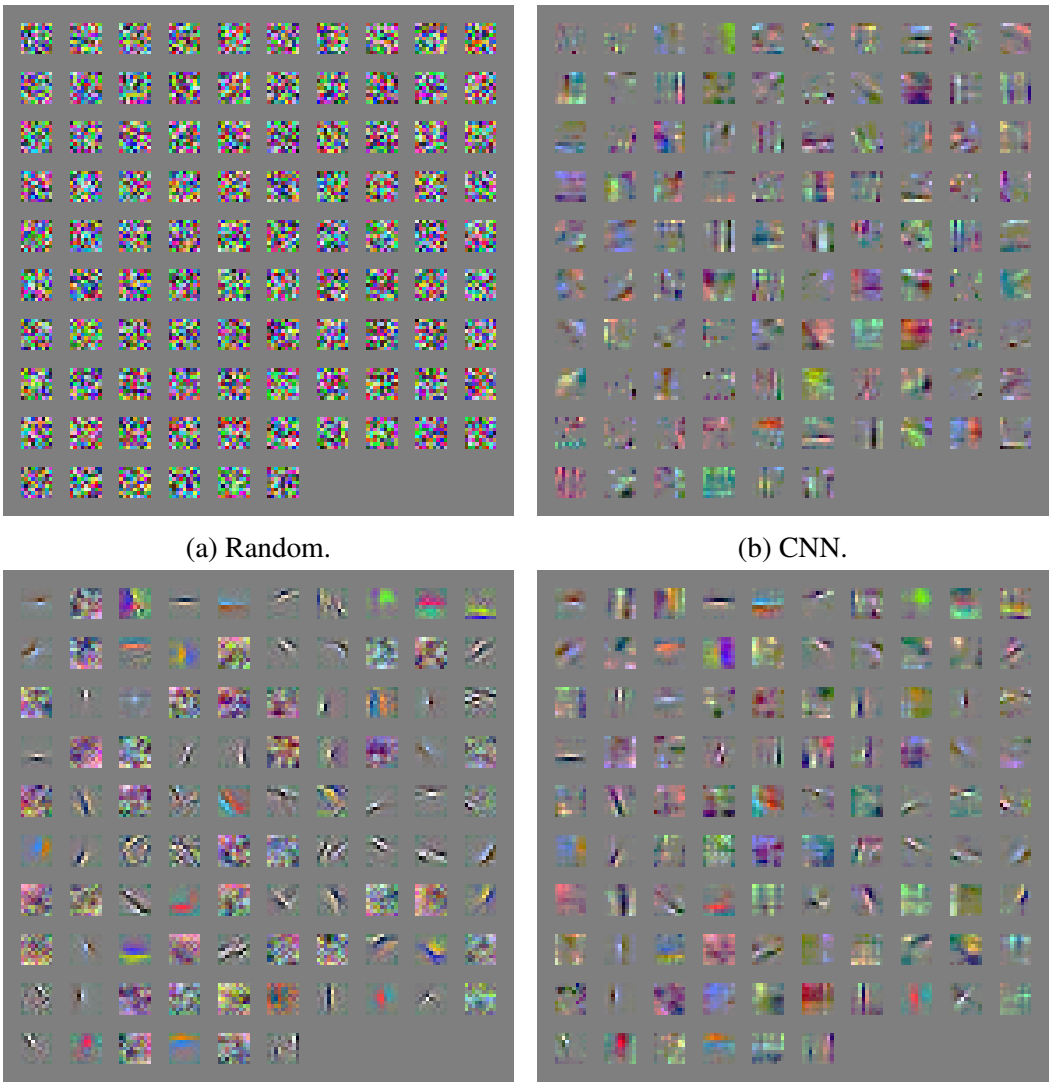
far from the state-of-the-art neural network method [54], which utilizes a model averaging technique by using committees of CNNs, it provides evidence that pretraining is helpful for convolutional networks trained on natural images. Standard deviations of two models were close to each other. The results are summarized in Table 4.

Table 4. Test accuracy for STL-10 dataset.

Model	Accuracy
ReLU-CNN	51.38% (0.41)
ReLU-CDBN	53.02% (0.39)
Committees of CNNs [54]	68% (0.55)

The progression of the first layer convolutional filters during supervised fine-tuning is visualized in Figure 17. Pretrained filters correspond to Gabor-like edge and color detectors. After fine-tuning, these filters are refined and color features become more dominant. By visually comparing CNN and CDBN filters, it is evident that pretraining becomes more important, when there is not much labeled data available in training.

Airplane, car and ship classes were classified significantly better than cat and dog classes. There were at least 38% difference in the classification accuracy between these classes. This is surprising because the classes in the training and the test set are evenly distributed. The full confusion matrix of CDBN model, averaged over the folds, is presented in Figure 18. We found using posterior distributions that only 6.56% of errors were corrected by the network's second guess. This is a much smaller number compared to the experiments on MNIST dataset. Together with classification results in general, it is a clear indication that natural images are a much harder problem for vision than handwritten digits.



(a) Random. (b) CNN.

(c) Pretrained. (d) CDBN.

Figure 17. First layer weights of CNN and CDBN learned on STL-10 dataset. Filters on the left column are used to initialize the supervised training step and the right column corresponds to fine-tuned filters.

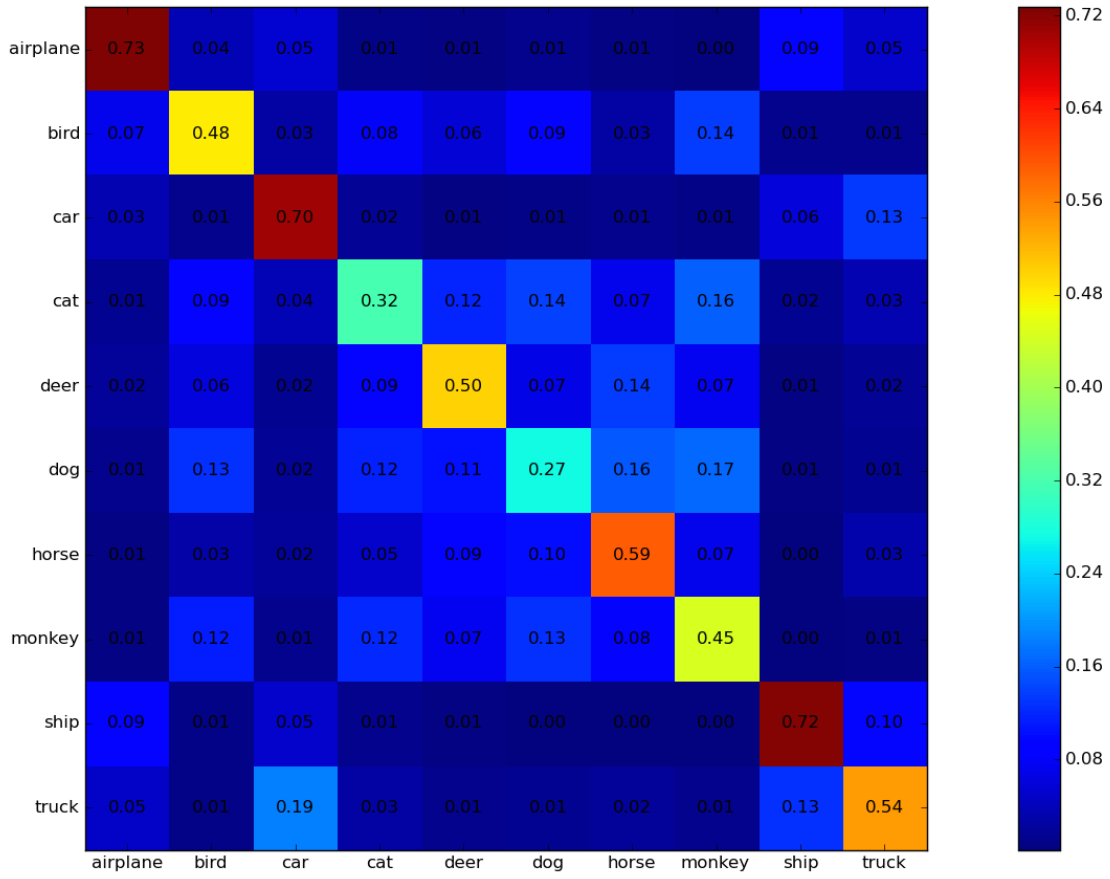


Figure 18. Confusion matrix on STL-10 dataset by our best model. Matrix is constructed by combining test predictions of all 10 CDBN models. Correct predictions are in the diagonal of the matrix.

5 DISCUSSION

5.1 Pretraining

In the first experiment, we tested the effect of generative pretraining with and without dropout. For the supervised step, we did not utilize any regularization or optimization tricks. We varied the size of the training set to see whether the effect of pretraining fades when more labeled data are available. Tests on MNIST dataset confirmed that the rectified linear unit does not suffer from the same optimization problems than sigmoid units. This can be seen from the relatively small difference between the baseline and pretrained models. Without other regularizations, pretraining was generally useful but only by a small margin, and the most importantly, it is not a required step for the convolutional network to learn.

Based on the MNIST results (Table 1.) we cannot say much about the usefulness of dropout during pretraining. However, qualitative analysis of the pretrained filters (Figure 11.) show that they contain less pen strokes on the initialization, corresponding to features with less structure. The convergence of pretrained filters were very similar, when larger training sets were used. In the case of 1,000 training images, dropout CDBN model got a significantly worse result than the one without dropout. One possible explanation can be that the added regularization was too strong for such a small amount of data. Great performance for dropout CDBN on the full dataset also supports this stronger regularization argument.

For natural images, zero-padding in the convolutional operation helped to achieve more centered filters (Figure 17c.), which are more general than cornered filters. The constant background in handwritten digits naturally introduced this zero-padding in the original images. Thus, for MNIST, we used full zero-padding only for the second layer. While there are some cornered filters in Figure 11, most of them correspond to centered pen strokes and point filters.

Committees of CNNs [54], the state-of-the-art method for STL-10 dataset, uses several neural network tricks such as local contrast normalization, random grouping for feature maps and building committees using two-level model averaging. They also used strided convolutions and full 96×96 images without subsampling. These advanced topics were considered as too broad for this thesis and were excluded from the scope. Also, our initial experiments with full-sized images without the strided convolution did not provide better

results and we continued experiments with subsampled images. This provided a fast way to achieve our initial goal to study the importance of pretraining and we fixed this setting in the early stage of experiments. All that being said, committees of CNNs did not utilize pretraining. This makes our findings on pretraining interesting because pretraining could be applied to Comittees of CNNs model, and increase the performance of the model even further.

5.2 Regularization and Performance Analysis

From the regularization experiments on handwritten digit classification, we found that a purely supervised model benefits more from the strong regularization and momentum optimization than a pretrained model. The implementations of data augmentation, dropout and momentum SGD are much easier and less error prone compared to pretraining. Omitting the pretraining step will also save the total training time. In practical applications, the convenience arguments are often much more important than a small gain in the accuracy. However, CNN with a strong regularization trained on the full MNIST dataset had larger error than the pretrained models without added regularization.

Our best model on the full MNIST training set, dropout CDBN, provided results comparable to the state-of-the-art. Inspecting errors of the model, in Figure 13, reveals, how useful for applications it is to have posterior probabilities. Even though the errors are quite evenly distributed among classes, some mistakes that are common for humans as well, can be found: 2 and 7 are confused; and 4 and 5 are classified as 9. Probably humans would not have done the same errors, but many of them are reasonable and were corrected by the network's second guess. The resulting accuracy would be sufficient for many applications, for example, reading postal codes from letters or reading numbers from vehicle registration plates.

Based on our initial experiments on STL-10, we decided to use full regularization. This was also backed by the experiments on MNIST: Even for a relatively simple dataset, the strong regularization and momentum optimization were beneficial for small training sets. Classifying natural images is a lot harder problem than the handwritten digits, and limiting the training set size makes it even harder. While this was expected, large interclass errors (Figure 18.) were surprising and the possibility to use the network's second guess provided very little practical gains. Both datasets have 10 classes. Error analysis suggests that in STL-10 dataset, the classes did not resemble each other that much compared to digits. The constant background in MNIST, for example, contains very little noise.

5.3 Future Work

Strided convolution should be tested to determine whether the architecture could exploit the full dimensionality of the 96×96 images on STL-10 dataset. More isolated tests for each regularization method would give comprehensive information on the studied methods. The proposed model, which utilizes generative pretraining, could be combined with the state-of-the-art model averaging method (Committees of CNNs [54]) on the STL-10 natural images classification task.

Recent studies on the properties of DNNs [15, 16] showed that adversarial examples and many other unrecognizable images can be used to cause prediction errors for networks that give good generalization performance on standard classification benchmarks. The smaller decision boundary created by a generative model, may become an important part on trying to alleviate this problem. This phenomenon is relevant in practical applications and comparing the proposed pretrained and purely supervised networks would be an interesting starting point for future research.

6 CONCLUSIONS

The goal of this study was to find out is the effect of pretraining in vision tasks damped by recent practical advances in optimization and regularization of Convolutional Neural Networks. The datasets of handwritten digits (MNIST) and natural images for developing unsupervised feature learning (STL-10) were used in experiments.

In this thesis a general introduction to Deep Neural Networks was given. We described fully connected and convolutional network architectures that can be trained using supervised backpropagation algorithm. Several regularization methods, such as generative pretraining, dropout, weight-decay and data augmentation, together with gradient-based momentum optimization, were introduced.

The proposed model with dropout pretraining provided 0.48% error on MNIST, which is comparable to the state-of-the-art methods. Analysis of the learned first layer filters show that with pretraining, the filters contain less noise when fine-tuned with smaller training set. For STL-10 dataset, the proposed pretraining method got 1.64% better results than the baseline. This provides evidence that pretraining is helpful for convolutional networks trained on natural images. Because STL-10 dataset contains very few labeled training examples, by visually inspecting trained filters it is evident that pretraining becomes more important.

The results of this work imply that pretraining is a substantial regularizer, however, not a necessary step in training Convolutional Neural Networks with rectified activations. Pretraining becomes more important when there is an insufficient amount of labeled data available. The proposed pretraining step can be included into the state-of-the-art model averaging method. Generative pretraining could also potentially mitigate the problem, where the predictions of purely discriminative networks are fooled by the adversarial examples (indistinguishable from regular examples by humans) and many other unrecognizable images.

REFERENCES

[1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. In *Neurocomputing: Foundations of Research*, pages 696–699. MIT Press, 1988.

[2] M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:76–86, 1992.

[3] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[4] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[5] ImageNet. `http://image-net.org/`. Accessed November 8, 2015.

[6] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814. Omnipress, 2010.

[7] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[8] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1319–1327. JMLR, Inc., 2013.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. volume 86, pages 2278–2324. IEEE, 1998.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). `http://www.image-net.org/challenges/LSVRC/2012/`. Accessed November 8, 2015.

[12] Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). `http://www.image-net.org/challenges/LSVRC/2014/`. Accessed November 8, 2015.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[14] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[16] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[18] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN*, volume 6354, pages 92–101. Springer, 2010.

[19] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[20] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[21] Kevin Jarrett, Koray Kavukcuoglu, M Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *IEEE Conference on International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

[22] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323. CSREA Press, 2011.

[23] John S Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems*, pages 211–217. Morgan Kaufmann, 1990.

[24] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[25] P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pages 194–281. MIT Press, 1986.

[26] Yann Lecun, Fu Jie, and Jhuangfu. Loss functions for discriminative training of energy-based models. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2005.

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

[28] KyungHyun Cho, Alexander Ilin, and Tapani Raiko. Improved learning of gaussian-bernoulli restricted boltzmann machines. In *Artificial Neural Networks and Machine Learning–ICANN*, pages 10–17. Springer, 2011.

[29] Nan Wang, Jan Melchior, and Laurenz Wiskott. An analysis of gaussian-binary restricted boltzmann machines for natural images. In *Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 287–292. Ciaco, 2012.

[30] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. In *Journal of Machine Learning Research*, pages 695–709. Microtome Publishing, 2005.

[31] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[32] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

[33] 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.

[34] 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*, 15:1929–1958, 2014.

[35] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[36] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

[37] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[38] Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *Learning Theory*, pages 545–560. Springer, 2005.

[39] Larry Yaeger, Richard Lyon, and Brandyn Webb. Effective training of a neural network character classifier for word recognition. In *Advances in neural information processing systems*, pages 807–816. MIT Press, 1996.

[40] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition*, volume 2, pages 958–958. IEEE Computer Society, 2003.

[41] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[42] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[43] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17:9, 1998.

[44] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[45] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147. JMLR, Inc., 2013.

[46] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`. Accessed November 8, 2015.

[47] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1058–1066. JMLR, Inc., 2013.

[48] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649. IEEE, 2012.

[49] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pages 2627–2635. Curran Associates, Inc., 2014.

[50] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223. JMLR, Inc., 2011.

[51] STL-10 dataset. `http://cs.stanford.edu/~acoates/stl10`. Accessed November 8, 2015.

[52] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012. Curran Associates, Inc., 2013.

[53] Anthony J Bell and Terrence J Sejnowski. Edges are the" independent components" of natural scenes. In *Advances in Neural Information Processing Systems*, pages 831–837. MIT Press, 1996.

[54] Bogdan Miclut. Committees of deep feedforward networks trained with few data. In *Pattern Recognition*, Lecture Notes in Computer Science, pages 736–742. Springer, 2014.