Prediction as a candidate for learning deep hierarchical models of data

Rasmus Berg Palm

Kongens Lyngby 2012
DTU-Informatic-Msc.-2012

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk DTU-Informatic-Msc.-2012

Abstract

Recent findings [HOT06] have made possible the learning of deep layered hierarchical representations of data mimicking the brains working. It is hoped that this paradigm will unlock some of the power of the brain and lead to advances towards true AI.

In this thesis I implement and evaluate state-of-the-art deep learning models and using these as building blocks I investigate the hypothesis that predicting the time-to-time sensory input is a good learning objective. I introduce the Predictive Encoder (PE) and show that a simple non-regularized learning rule, minimizing prediction error on natural video patches leads to receptive fields similar to those found in Macaque monkey visual area V1. I scale this model to video of natural scenes by introducing the Convolutional Predictive Encoder (CPE) and show similar results. Both models can be used in deep architectures as a deep learning module.

Preface

This thesis was prepared at DTU Informatics at the Technical University of Denmark in fulfilment of the requirements for acquiring a M.Sc. in Medicine & Technology.

Lyngby, 28-March-2012

Rasmus Berg Palm

Acknowledgements

I would like to thank my main supervisor Ole Winter for insightful suggestions and guidance, and I would like to thank my secondary supervisor Morten Mørup for his keen insights and tireless assistance.

Contents

Introduction

The neocortex is the most powerful learning machine we know of to date. While computers clearly outperform humans on raw computational tasks, no computer can learn to speak a language, make a sandwich and navigate our homes. One might, after considerable effort create a sandwich making robot, but it would fail horribly at learning a language. The most remarkable ability of the neocortex is its ability to handle a great number of different tasks: sensing, motor control, and higher-level cognition such as planning, execution, social navigation, etc.

If this exceedingly diverse and complex behaviour is the result of billions of years of hard-coded evolution with no inherit structure, then it would be nearly impossible to reverse engineer and we should not be looking for principles, but rather accept the intricate beauty of our connectome. But, to the contrary the neocortex:

- learns - language, motor control, etc. are not skills humans are born with, they are learned.
- is plastic to such an extent that one neocortical area can take over another areas function [BM98, Cre77]
- is a highly repetitive structure built of billions of nearly identical columns [Mou97].
- is layered and hierarchical such that each layer learns more abstract concepts [FVE91].

This leads us to believe that the neocortex is built using a general purpose learning algorithm and a general purpose architecture. It seems then, that the complexity of the neocortex is limited to a single learning module of a more manageable size and the hope is that if we can understand this module, we can replicate it and apply it in a scale that leads to true artificial intelligence.

Jeff Hawkins proposed that the general neocortical algorithm is a prediction algorithm, and that learning is optimizing prediction[HB04]. This is in line with Karl Friston's 2003 unifying brain theory stating that the brain seeks to minimize free energy, or prediction error[Fri03] and is generally in line with a large base of literature on predictive coding [SLD82] [RB99] [Spr12] which states that top down connections seeks to predict lower layer activities and amplify those consistent with the predictions. In close proximity to this direct prediction learning paradigm is the observation of temporal coherence. Temporal coherence is the observation that our rapidly changing sensory input is a highly non-linear combination of slowly changing high-level objects and features. Several methods have been proposed to extract these high-level features which are changing slowly over time [BW05] [MMC+09]. In the same line of reasoning it has been proposed that one should measure the performance of a model by its invariance to temporally occurring transformations [GLS+09]. While several models have been proposed I feel that the simple basic idea that prediction error drives learning have not been sufficiently tested. The goal of this thesis is to propose, implement and evaluate a prediction learning algorithm.

In his seminal paper[HOT06], Hinton essentially created the field of deep learning by showing that learning deep, layered hierarchical models of data was possible using a simple principle and that these deep architectures had superior performance to state-of-the-art machine learning models. Hinton showed that instead of training a deep model directly towards optimizing performance on a supervised task, e.g. recognizing hand written digits, one should train layers of a deep model individually on an unsupervised task; to represent the input data in an 'good' way. The lowest level layer would learn to represent images of hand written digits, and the layer above would learn to represent the representation of the layer below. Having stacked multiple layers like this the combined model could finally be trained on the supervised task using these new higher level representation of the data which lead to superior performance. The definition of what comprises a 'good' representation of data and how it is learned is to a large degree the subject of research in deep learning. The deep learning paradigm of unsupervised learning of layered hierarchical models of unstructured data is similar to the previously described architecture and working of the neocortex making it a good choice of theoretical and computational framework for implementing and evaluating the prediction learning algorithm.

The first chapter will motivate deep learning and implement and evaluate the basic building block of deep learning models in order to gain an insight into how and why they work. The second chapter will re-iterate the case for a prediction learning framework and propose, implement and evaluate the proposed methods using the lessons learned from chapter one. The thesis ends with a discussion of the findings, the problems encountered, and proposes possible solutions and future research.

Chapter 1

Deep Learning

1.1 Introduction

1.1.1 What is Deep Learning?

Deep Learning is a machine learning paradigm that focuses on learning deep hierarchical models of data. Deep Learning hypothesizes that in order to learn high-level representations of data a hierarchy of intermediate representations are needed. In the vision case the first level of representation could be gabor-like filters, the second level could be line and corner detectors, and higher level representations could be objects and concepts. Recent advances in learning algorithms for deep architectures [HOT06] has made deep learning feasible and deep learning systems have since beat or achieved state-of-the-art performance on numerous machine learning tasks [HOT06, MH10, LZYN11, VLL+10].

Deep Learning is most easily explained in contrast to more shallow learning methods. An archetypical shallow learning methods might be a feedforward neural network with an input layer, a single hidden layer and an output layer trained with backpropagation on a classification task.

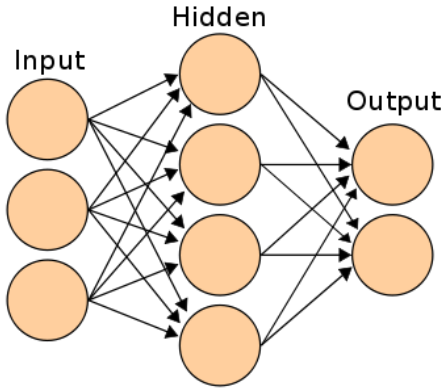

Figure 1.1: Shallow Feed Forward Neural Net (1 hidden layer).

Best practises for neural network suggests that adding more hidden layers than one or two is rarely a good idea[dVB93]. Instead one can increase the hidden layers width as it has been shown that enough hidden units can approximate any function[HSW89]. Due to the shallow architecture, each hidden neuron is forced to represent something that can be immediately used for classification. If the classification task is to distinguish between pictures of cats and dogs a hidden neuron could represent 'dog paw' or 'cat paw' but not the common feature 'paw'. This is an oversimplification, as the output layer provides a last level of combination and evaluation of features, but the point remains: In a feedforward neural net of N layers, there are at most N possibilities to combine lower level features.
The Deep Learning equivalent would be a feedforward neural network with many hidden layers. Many in this context being 3 or more. The theory is that if the neural net is allowed to find meaningful representations on several levels it will perform better. The first hidden level could represent edges or strokes, the second would be combinations of edges/strokes, i.e. corners/circles, and so on, each layer seeing patterns in lower levels and representing more and more abstract concepts. This seems like a good idea in theory, but early neural net pioneers found that it was not as easy as merely piling on more layers, which lead to the previously described best practices.
The preceding example used neural networks as the learning module, but the general principle in Deep Learning is that learning modules should be applied recursively, each time finding more complex patterns.
The field of Deep Learning studies these learning modules. Which modules work? How do we measure their performance? How do we train them?

1.1.2 Motivations for Deep Learning

1.1.2.1 Biological motivations

A key motivation for deep learning is that the brain seems to operate in a 'deep' fashion, more specifically, the neocortex has a number of attributes which speaks in favour of investigating deep learning.

One of Deep Learnings most important neocortical motivations is that the neocortex is layered and hierarchical. Specifically it has approximately 6 layers [KSJ00] with lower layers projecting to higher layers and higher layers projecting back to lower layers[GHB07][EGHP98]. The hierarchical nature comes from the observation that generally the upper layers represents increasingly abstract concepts and are increasingly invariant to transformations such as lighting, pose, etc. The archetypical example is the visual pathway in which it was found that V1, taking input from the sensory cells, reacted the strongest to simple inputs modelled very well by gabor filter[HW62][Dau85]. As information flows from V1 to the higher areas V2 and V4 and IT the neurons become responsive to increasingly abstract features and observe increased invariances to viewpoint, lighting, etc. [QvdH05] [GCR+96] [BTS+01].

Deep learning believes that this deep, hierarchical structure employed by the neocortex is the answer to much of its power and attempts to unlock these powers by examining the effects of layered and hierachial structures.

It should be noted that cognitive psychologists have examined the idea of a layered hierarchical computational brain model for a long time. The Deep Learning field borrows a lot from these earlier ideas and can be seen as trying to implement some of these ideas [Ben09].

1.1.2.2 Computational Power

An important theoretical motivation for studying Deep Learning is that in some cases a computation that can be achieved efficiently with k layers is exponentially more inefficiently computed with k-1 layers [Ben09]. In this case efficiently refers to the number of computational elements required to perform the computation. In neural networks, a computational element would be a neuron, and in a logical circuit it would an AND, OR, XOR, etc gate. Layers refers to the longest number of computational steps to get output from input. In a computational

network with the multiplication, subtraction, addition and the sin operators expressing $f(x) = x*\sin(a*x+b)$ would require the longest chain to be 4 steps long

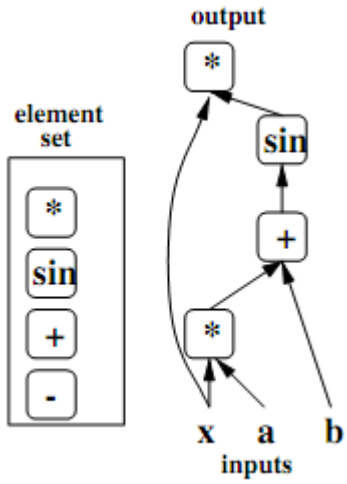

Figure 1.2: Computational graph implementing $x * \sin(a * x + b)$. Taken from [Ben09]

Computational efficiency, or compactness, matters in machine learning as the parameters of the computational elements are usually what is learned during training, and the training set size is usually fixed. As such, additional computational elements represents more parameters per training examples, resulting in worse performance. Further, when adding additional parameters, there is always a risk of over-fitting leading to poor generalization.

Formal mathematical proof of the computational in-effciency of k-1 deep architectures compared to k deep architectures exists in some cases of network architecture [Has86], however it remains intuition that this applies to the kinds of networks and functions typically applied in deep learning[Ben09].

An example illustrating the phenomenon is trying to fit a third order sinusoid, $f(x) = \sin(\sin(\sin(x)))$ in a neural net architecture with neurons applying the sinusoid as their non-linearity.

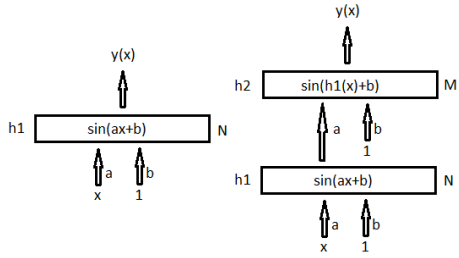

Figure 1.3: One (left) and two layer (right) computational models.

In a model with an input layer x, a single hidden layer, $h1$ and an output layer y we get the following.

$$h1_n(x) = \sin(b_n^{(1)} + a_n^{(1)} * x) \tag{1.1}$$

$$y(x) = \sin(b^{(2)} + \sum_{n=1}^{N} a_n^{(2)} * h1_n(x)) \tag{1.2}$$

$$L(x, a, b) = (y(x) - f(x))^2 \tag{1.3}$$

Where N is the number of units in the hidden layer, a and b are a multiplicative and additive parameter respectively, the superscripts indicate the layer the parameters are associated with and L is the loss function we are trying to minimize. It can be seen that the single hidden layer model would not be able to fit the third order sinusoid perfectly and that its precision would increase with N, the width of the hidden layer.
Compared to a network with two hidden layers, the second hidden layer having M units:

$$h1_n(x) = \sin(b_n^{(1)} + a_n^{(1)} * x) \tag{1.4}$$

$$h2_m(x) = \sin(b_m^{(2)} + \sum_{n=1}^{N} a_n^{(2)} m * h1_n(x)) \tag{1.5}$$

$$y(x) = \sin(b^{(3)} + \sum_{m=1}^{M} a_m^{(3)} * h_m(x)) \tag{1.6}$$

$$L(x, a, b) = (y(x) - f(x))^2 \tag{1.7}$$

It is evident that the network with two hidden layers would fit the third order sinusoid perfectly with $N = M = 1, a_1^{(1)} = a_{11}^{(2)} = a_1^{(3)} = 1, b_1^{(1)} = b_1^{(2)} = b_1^{(3)} = 0$, i.e just 1 unit in both hidden layers and just 6 parameters (three of them being zero).

1.2 Methods

1.2.1 Deep Learning Primitives

In the following, the three most prominent deep learning primitives will be described in some detail in their simplest form. These primitives or building blocks are at the foundation of many deep learning methods and understanding their basic form will allow the reader to quickly understand more complex models relying on these building blocks.

1.2.1.1 Deep Belief Networks

Deep Belief Networks (DBNs) consists of a number of layers of Restricted Boltzmann Machines (RBMs) which are trained in a greedy layer wise fashion. A RBM is an generative undirected graphical model.

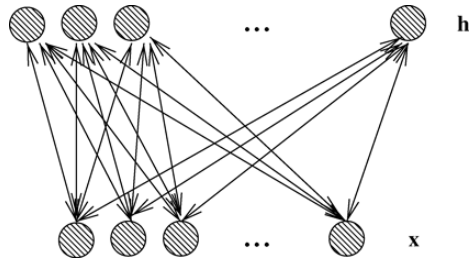

Figure 1.4: Restricted Boltzmann Machine. Taken from [Ben09].

The lower layer x, is defined as the visible layer, and the top layer h as the hidden layer. The visible and hidden layer units x and h are stochastic binary variables. The weights between the visible layer and the hidden layer are undirected and are denoted W. In addition each neuron has a bias. The model defines the probability distribution

$$P(x,h) = \frac{e^{-E(x,h)}}{Z} \tag{1.8}$$

With the energy function, $E(x, h)$ and the partition function Z being defined as

$$E(x, h) = -b'x - c'h - h'Wx \tag{1.9}$$

$$Z(x, h) = \sum_{x,h} e^{-E(x,h)} \tag{1.10}$$

Where b and c are the biases of the visible layer and the hidden layer respectively. The sum over x, h represents all possible states of the model.
The conditional probability of one layer, given the other is

$$P(h|x) = \frac{\exp(b'x + c'h + h'Wx)}{\sum_h \exp(b'x + c'h + h'Wx)} \tag{1.11}$$

$$P(h|x) = \frac{\prod_i \exp(c_i h_i + h_i W_i x)}{\prod_i \sum_h \exp(c_i h_i + h_i W_i x)} \tag{1.12}$$

$$P(h|x) = \prod_i \frac{\exp(h_i(c_i + W_i x))}{\sum_h \exp(h_i(c_i + W_i x))} \tag{1.13}$$

$$P(h|x) = \prod_i P(h_i|x) \tag{1.14}$$

Notice that if one layer is given, the distribution of the other layer is factorial. Since the neurons are binary the probability of a single neuron being on is given by

$$P(h_i = 1|x) = \frac{\exp(c_i + W_i x)}{1 + \exp(c_i + W_i x)} \tag{1.15}$$

$$P(h_i = 1|x) = \text{sigm}(c_i + W_i x) \tag{1.16}$$

Similarly the conditional probability for the visible layer can be found

$$P(x_i = 1|h) = \text{sigm}(b_i + W_i h) \tag{1.17}$$

In other words, it is a probabilistic version of the normal sigmoid neuron activation function. To train the model, the idea is to make the model generate data like the training data. Mathematically speaking we wish to maximize the log probability of the training data or minimize the negative log probability of the training data.
The gradient of the negative log probability of the visible layer with respect to

the model parameters θ is

$$\begin{aligned}
&\frac{\partial}{\partial\theta}(-\log P(x)) = \frac{\partial}{\partial\theta}\left(-\log\sum_h P(x,h)\right)\\
&= \frac{\partial}{\partial\theta}\left(-\log\sum_h \frac{\exp(-E(x,h))}{Z}\right)\\
&= -\frac{Z}{\sum_h \exp(-E(x,h))}\left(\sum_h \frac{1}{Z}\frac{\partial\exp(-E(x,h))}{\partial\theta} - \sum_h \frac{\exp(-E(x,h))}{Z^2}\frac{\partial Z}{\partial\theta}\right)\\
&= \sum_h\left(\frac{\exp(-E(x,h))}{\sum_{\hat{h}}\exp(-E(x,\hat{h}))}\frac{\partial E(x,h)}{\partial\theta}\right) + \frac{1}{Z}\frac{\partial Z}{\partial\theta}\\
&= \sum_h P(h|x)\frac{\partial E(x,h)}{\partial\theta} - \frac{1}{Z}\sum_{x,h}\exp(-E(x,h))\frac{\partial E(x,h)}{\partial\theta}\\
&= \sum_h P(h|x)\frac{\partial E(x,h)}{\partial\theta} - \sum_{x,h} P(x,h)\frac{\partial E(x,h)}{\partial\theta}\\
&= \mu_1\left[\frac{\partial E(x,h)}{\partial\theta}\middle|\, x\right] - \mu_1\left[\frac{\partial E(x,h)}{\partial\theta}\right]\\
&\frac{\partial}{\partial W}(-\log P(x)) = \mu_1\left[-h'x|\, x\right] - \mu_1\left[-h'x\right]\\
&\frac{\partial}{\partial b}(-\log P(x)) = \mu_1\left[-x|\, x\right] - \mu_1\left[-x\right]\\
&\frac{\partial}{\partial c}(-\log P(x)) = \mu_1\left[-h|\, x\right] - \mu_1\left[-h\right]
\end{aligned}$$

where μ_1 is a function returning the first moment or expected value. The first contribution is dubbed the positive phase, and it lowers the energy of the training data, the second contribution is dubbed the negative phase and it raises the energy of all other visible states the model is likely to generate.
The positive phase is easy to compute as the hidden layer is factorial given the visible layer. The negative phase on the other hand is not trivial to compute as it involves summing all possible states of the model.
Instead of computing the exact negative phase, we will sample from the model. Getting samples from the model is easy; given some state of the visible layer, update the hidden layer, given that state, update the visible layer, and so on,

i.e.

$$
\begin{aligned}
h^{(0)} &= P(h|x^{(0)}) \\
x^{(1)} &= P(x|h^{(0)}) \\
h^{(1)} &= P(h|x^{(1)}) \\
&... \\
x^{(n)} &= P(x|h^{(n-1)})
\end{aligned}
$$

The superscripts denote the order in which each calculation is made, not the specific neuron of the layer. At each iteration the entire layer is updated. To get unbiased samples, we should initialize the model at some arbitrary state, and sample n times, n being a large number. To make this efficient, we'll do something slightly different. We'll initialize the model at a training sample, iterate one step, and use this as our negative sample. This is the contrastive divergence algorithm as introduced by Hinton [HOT06] with one step (CD-1). The logic is that, as the model distribution approaches the training data distribution, initializing the model to a training sample approximates letting the model converge.
Finally, for computational efficiency, we will use stochastic gradient descent instead of the batch update rule derived. Alternatively one can use mini-batches. The final RBM learning algorithm can be seen below. α is a learning rate and $rand()$ produces random uniform numbers between 0 and 1.

Algorithm 1 Contrastive Divergence 1

for all training samples as t **do**
  $x^{(0)} \leftarrow t$
  $h^{(0)} \leftarrow sigm(x^{(0)}W + c) > rand()$
  $x^{(1)} \leftarrow sigm(h^{(0)}W^T + b) > rand()$
  $h^{(1)} \leftarrow sigm(x^{(1)}W + c) > rand()$
  $W \leftarrow W + \alpha(x^{(0)}h^{(0)} - x^{(1)}h^{(1)})$
  $b \leftarrow b + \alpha(x^{(0)} - x^{(1)})$
  $c \leftarrow c + \alpha(h^{(0)} - h^{(1)})$
end for

Being able to train RBMs we now turn to putting them together to form deep belief networks. The idea is to train the first RBM as described above, then train another RBM using the first RBM's hidden layer as the second RBMs visible layer.

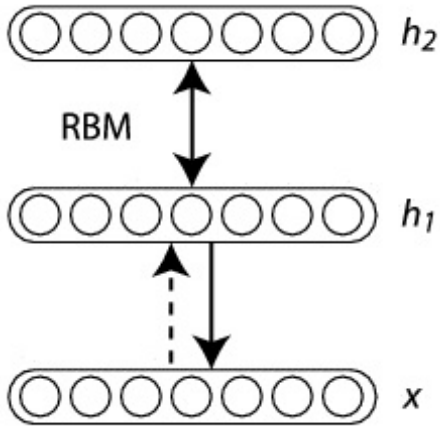

Figure 1.5: Deep Belief Network. Taken from [Ben09].

To train the second RBM, a training sample is clamped to x, transformed to h_1 by the first RBM and then contrastive divergence is used to train the second RBM. As such training the second RBM is exactly equal to training the first RBM, except that the training data is mapped through the first RBM before being used as training samples. The intuition is that if the RBM is a general method for extracting a meaningful representation of data, then it should be indifferent to what data it is applied to. Popularly speaking, the RBM doesn't know whether the visible layer is pixels, or the output of another RBM, or something different altogether. With this intuition it becomes interesting to add a second RBM, to see if it can extract a higher level representation of the data. Hinton et al have shown, that adding a second RBM decreases a variational band on the log likelihood of the training data [HOT06].
Having trained N layers in this unsupervised greedy manner, Hinton et al, adds a last RBM and adds a number of softmax neurons to its visible layer. The softmax neurons are then clamped to the labels of the training data, such that they are 0 for all except the neuron corresponding to the label of the training sample, which is set to 1. In this way, the last RBM learns a joint model of the transformed data, and the labels.

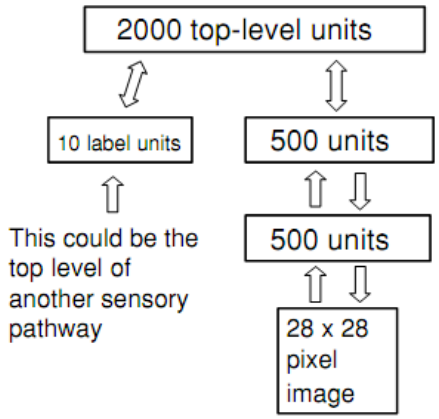

Figure 1.6: Deep Belief Network with softmax label layer. Taken from [HOT06].

To use the DBN for classification, a sample is clamped to the lowest level visible layer, transformed upwards through the DBN until it reaches the last RBMs hidden layer. In these upward passes the probabilities of hidden units being on are used directly instead of sampling, to reduce noise. At the top RBM, a few iterations of gibbs sampling is performed after which the label is read out. Alternatively the exact 'free energy' of each label can be computed and the one with the lowest free energy is chosen [HOT06]. To fine-tune the entire model for classification Hinton et al uses an 'up-down' algorithm [HOT06].
Simpler ways to use the DBN for classification are to simply use the top level RBM hidden layer activation in any standard classifier or to add a last label layer, and train the whole model as a feedforward-backpropagate neural network. If one of the latter methods are used, then there is no need to train the last RBM as a joint model of data and labels.

1.2.1.2 Stacked Autoencoders

Stacked Autoencoders are, as the name suggests, autoencoders stacked on top of each other, and trained in a layerwise greedy fashion.
An autoencoder or auto-associator is a discriminative graphical model that attempts to reconstruct its input signals.

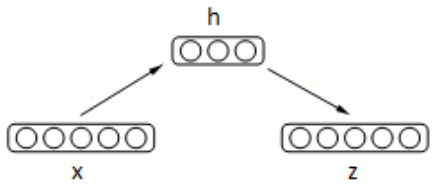

Figure 1.7: Autoencoder. Taken from [Ben09].

There exists a plethora of proposed autoencoder architectures; with/without tied weights, with various activation functions, with deterministic/stochastic variables, etc. This section will use the autoencoder desribed in [BLP+07] as a basic example implementation which have been used successfully as a building block for deep architectures.
Autoencoders take a vector input x, encodes it to a hidden layer h, and decodes it to a reconstruction z.

$$h(x) = \text{sigm}(W_1 x + b_1) \tag{1.18}$$

$$z(x) = \text{sigm}(W_2 h(x) + b_2) \tag{1.19}$$

To train the model, the idea is to minimize the average difference between the input x and the reconstruction z with respect to the parameters, here denoted θ.

$$\theta = \underset{\theta}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} L(x^{(i)}, z(x^{(i)})) \tag{1.20}$$

Where N is the number of training samples and L is a function that measures the difference between x and z, such as the traditional squared error $L(x, z) = ||x - z||^2$ or if x and z are bit vectors or bit probabilities, the cross-entropy $L(x, z) = x^T log(z) + (1 - x)^T log(1 - z)$
Updating the parameters efficiently is achieved with stochastic gradient descent which can be efficiently implementing using the backpropagation algorithm.

There is a serious problem with autoencoders, in that if the hidden layer is the same size or greater than the input and reconstruction layers, then the algorithm could simply learn the identity function. If the hidden layer is smaller than the input layer, or if other restrictions are put on its representation, e.g. sparseness, then this is not an issue.

Having trained the bottom layer autoencoder on data, a second layer autoencoder can be trained on the activities of the first autoencoders hidden layer when

exposed to data. In other words the second autoencoder is trained on $h^{(1)}(x)$ and the third autoencoder would be trained on $h^{(2)}(h^{(1)}(x))$, etc, where the superscripts denote the layer of the autoencoder. In this way multiple autoencoders can be stacked on top of each other and trained in a layer-wise greedy fashion, which has been shown to lead to good results [VLBM08].

To use the model for discrimination, the outputs of the last layer can be used in any standard classifier. Alternatively a last supervised layer can be added, and the whole model trained as a feedforward-backpropagate neural network.

1.2.1.3 Convolutional Neural Nets

Convolutional Neural Networks (CNNs) are feedforward, backpropagate neural networks with a special architecture inspired from the visual system. Hubel and Wiesel's early work on cat and monkey visual cortex showed that the visual cortex is composed of cells with high specificity to patterns within a localized area, called their receptive fields [HW68]. These so called simple cells are tiled as to cover the entire visual field and higher level cells recieve input from these simple cells, thus having greater receptive fields and showing greater invariance to translation. To mimick these properties Yan Lecun introduced the Convolutional Neural Network [LCBD$^+$90], which still hold state-of-the art performance on numerous machine vision tasks [CMS12] and acts as inspiration to recent reseach [SWB$^+$07], [LGRN09].

CNNs work on the 2 dimensional data, so called maps, directly, unlike normal neural networks which would concatenate these into vectors. CNNs consists of alternating layers of convolution layers and sub-sampling/pooling layers. The convolution layers compose feature maps by convolving kernels over feature maps in layers below them. The subsampling layers simply downsample the feature maps by a constant factor.

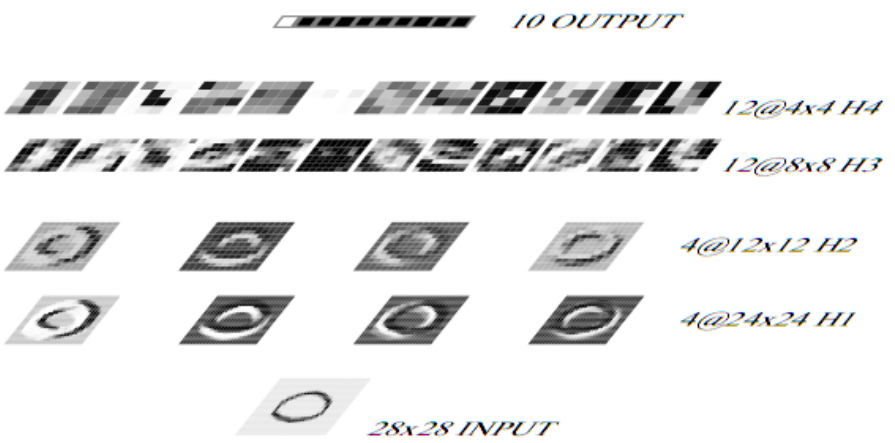

Figure 1.8: Convolutional Neural Net. Taken from [LCBD+90].

The activations a_j^l of a single feature map j in a convolution layer l is

$$a_j^l = f\left(b_j^l + \sum_{i \in M_j^l} a_i^{l-1} * k_{ij}^l\right) \tag{1.21}$$

Where f is a non-linearity, typically tanh() or sigm(), b_j^l is a scalar bias, M_j^l is a vector of indexes of the feature maps i in layer $l-1$ which feature map j in layer l should sum over, $*$ is the 2 dimensional convolution operator and k_{ij}^l is the kernel used on feature map i in layer $l-1$ to produce input to the sum in feature map j in layer l.
For a single feature map j in a subsampling layer l

$$a_j^l = \text{down}(a_j^{l-1}, N^l) \tag{1.22}$$

Where down is a function that downsamples by a factor N^l. A typical choice of down-sampling is mean-sampling in which the mean over non-overlapping regions of size $N^l \text{x} N^l$ are calculated.
To discriminate between C classes a fully connected output layer with C neurons are added. The output layer takes as input the concatenated feature maps of the layer below it, denoted the feature vector, fv

$$o = f\left(b^o + W^o fv\right)) \tag{1.23}$$

Where b^o is a bias vector and W^o is a weight matrix. The learnable parameters of the model are k_{ij}^l, b_j^l, b^o and W^o. Learning is done using gradient descent which

can be implemented efficiently using a convolutional implementation of the back-propagation algorithm as shown in [Bou06]. It should be clear that because kernels are applied over entire input maps, there are many more connections in the model than weights, i.e. the weights are shared. This makes learning deep models easier, as compared to normal feedforward-backprop neural nets, as there are fewer parameters, and the error gradients goes to zero slower because each weight has greater influence on the final output.

1.2.1.4 Sparsity

Sparse coding is the paradigm that data should be represented by a small subset of available basis functions at any time, and is based on the observation that the brain seems to represent information with a small number of neurons at any given time [OF04]. Sparsity was originally investigated in the context of efficient coding and compressed sensing and was shown to lead to gabor-like filters [OF97]. They are not directly related to deep architectures, but their interesting encoding properties have lead to them being used in deep learning algorithms in various ways.

The most direct use of sparse coding can be seen as formulating a new basis for a dataset which is composed of a feature vector and a set of basis functions, while restricting the feature vector to be sparse.

$$x = Aa \tag{1.24}$$

Where $x \in \Re^n$ is the data, $A \in \Re^{nxm}$ is the m basis functions and $a \in \Re^m$ is the "sparse" vector describing which sum of basis functions represent the data. a is sparse in the sense that it is mostly zero, i.e. few basis functions are used at all times to represent any data. In images this is in contrast to the normal non-sparse representation used, which is pixel intensities. This corresponds to $A = I^{nxn}$, i.e A being a square identity matrix, and a being the normal vector representation of image intensities.

In a deep learning setting, a non-sparsity penalty, i.e. measuring how much the neurons are active on average, can be added to the loss function. If the neurons are binomial, sparse coding could correspond to restricting the mean number of activate hidden neurons to some fraction. If the neurons are continuous valued, this could correspond to restricting the mean of all hidden units to some constant. Sparse variants of RBMs and autoencoders have been proposed

[GLS+09, PCL06]. A sparse autoencoder has a second contribution to its loss function, the non-sparsity measure:

$$L(x, z) = ||x - z||^2 + \beta||h - \rho|| \tag{1.25}$$

Where β is a constant describing how much being non-sparse should be penalized and ρ is a constant close to the lower boundary of the output of the hidden neurons h. In the case of sigmoidal hidden neurons, this could be $\rho = 0.1$. In the case of tanh hidden neurons this could be $\rho = -0.9$

1.2.2 Key Points

1.2.2.1 Training Deep Architectures

A key element to Deep Learning is the ability to learn from unlabelled data as it is available in vast quantities, e.g. video or images of natural scenes, sound, etc. This is also referred to as unsupervised training. This is in contrast to supervised training which is training on labelled data, of which there is relatively little, and which is much harder to generate. To get unlabelled video data one might simply go onto youtube.com and download a million hours of video, but to get 1 hour of labelled video data one would need to painstakingly segment each frame into the objects of interest. Further, being able to learn on the unlabelled data gives one a high-dimensional learning signal, whereas most labelled data is relatively low-dimensional, e.g. a label specifying cat or dog is two bits of information whereas a 100x100 pixels image of a cat or dog in true color is 24*3*100*100 = 720.000 bits.

Machine Learning models are rarely built to achieve good performance on unlabelled data though; usually some kind of classification or regression is required. The idea then is to train the model on the unlabelled data first, called pre-training, to achieve good features or representations of the data. Once this has been achieved the parameters learned are used to initiate a model, which is trained in a supervised fashion to fine-tunes the parameters to the task at hand.

Deep belief nets and stacked auto encoders both use the same method for pre-training: training each layer unsupervised on the activations of the layer below, one after another. Convolutional neural nets stand out in this aspect, as they do not use pre-training. Since CNNs have substantially less parameters, and translation invariance is built into the model, there seems to be less need for pre-training. Pre-training convolutional neural nets in a layer-wise fashion similar

to DBNs and SAEs have been shown to be slightly superior to randomly initialized networks though [MMCS11]. Generally the paradigm of greedy layer-wise pre-training followed by global supervised training to fine-tune the parameters seems to give good results.

The theory as to why this works well is that the unsupervised pre-training moves the parameters to a region in parameter space that is closer to a global optimum or at least a region which represents the data more naturally. Numerical studies have shown that pre-trained and randomly initialized networks do indeed end up in very different regions of parameter space after having been trained on a supervised task [EBC+10]. Also, the global supervised learning rarely changes the pre-trained parameters much, what happens instead is a fine-tuning to improve on the supervised task [EBC+10].

After pre-training, instead of training the model globally on the supervised task one can instead use any standard supervised learning model on the output features of the pre-trained model, e.g. pre-training a Deep Belief Network, and then using the activity of the top output neurons as input in a SVM, logistic regression, etc.

Alternatively one can train the model in a supervised and unsupervised setting at the same time, alternating between the two learning modes or having a composite learning rule. This is known as semi-supervised learning.

1.3 Results

The three primitives, DBNs, SAEs and CNNs were implemented and evaluated on the MNIST dataset to illustrate state of the art in Deep Learning. The error rates achieved for the DBN, SAE and CNN were 1.67%, 1.71% and 1.22%, respectively. The error rates compared to state-of-the art with comparable network architectures for the DBN and SDAEs are slightly worse whereas the CNN error rate is slightly better.

DBN (1000-1000-1000)	SAE (1000-1000-1000)	CNN (c6-d2-c72-d2)
1.67 %	1.71 %	1.22 %

Table 1.1: Error rates of three deep learning primitives. The DBN and SAE both had 3 hidden layers each with 1000 neurons. The CNN had the following layers: 6 feature maps using 5x5 kernels, 2x2 mean-pool downsampling, 72 feature maps using 5x5 kernels, 2x2 downsampling and a fully connected output layer.

The MNIST dataset [LBBH98] contains 70.000 labelled images of handwritten digits. The images are 28 by 28 pixels and gray scale. The dataset is divided into a training set of 60.000 images and a test set of 10.000 images. The dataset has been widely used as a benchmark of machine learning algorithms. In the following details of the implementations of the three models on MNIST is described and results on MNIST are shown.

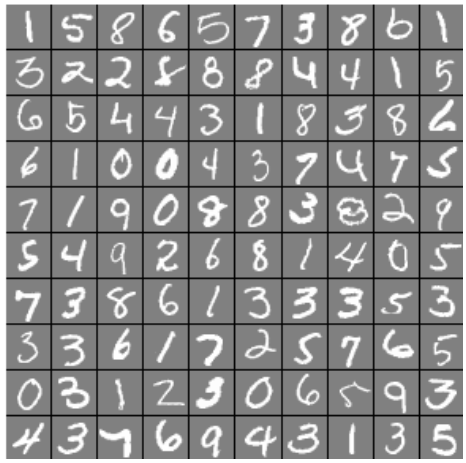

Figure 1.9: A random selection of MNIST training images.

Except otherwise noted all experiments used the sigmoid non-linearity for all neurons, initialized the biases to zero and drew weights from a uniform random distribution with upper and lower bounds $\pm\sqrt{6/(fan_{in} + fan_{out})}$ as recommended in [LBOM98]. All experiments were run on a machine with a 2.66 GHz Intel Xeon Processor and 24 GB of memory.

1.3.1 Deep Belief Network

A three layer DBN were constructed. The net consisted of three RBMs each with 1000 hidden neurons, and each RBM was trained in a layer-wise greedy manner with contrastive divergence. All weights and biases were initialized to be zero. Each RBM was trained on the full 60.000 images training set, using mini-batches of size 10, with a fixed learning rate of 0.01 for 100 epochs. One epoch is one full sweep of the data. The mini-batches were randomly selected each epoch. Having trained the first RBM the entire training dataset was transformed through the first RBM resulting in a new 60.000 x 1000 dataset which the second RBM was trained on and similarly so for the third RBM. Having pre-trained each RBM the weights and biases were used to initialize a feed-forward neural net with 4 layers of sizes 1000-1000-1000-10, the last 10 neurons being the output label units. The FFNN was trained using mini-batches of size 10 for 50 epochs using a fixed learning rate of 0.1 and a small L2 weight-decay of 0.00001 using back-propagation. To evaluate the performance the test set was feed-forwarded and the maximum output unit was chosen as the label for each sample resulting in an error rate of 1.67% or 167 errors out of the 10.000 test samples. The code ran for 28 hours.

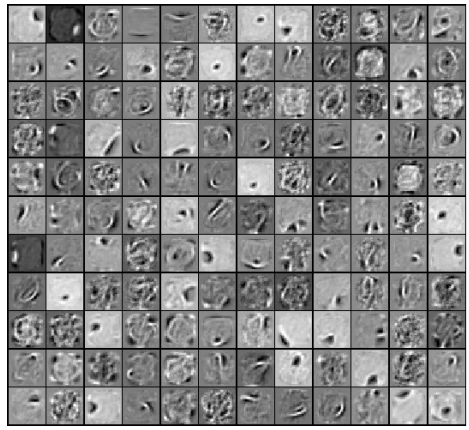

Figure 1.10: Weights of a random subset of the 1000 neurons of the first RBM. Each image is contrast normalized individually to be between minus one and one.

The first RBM has to a large degree learned stroke and blob detectors as can be seen from the weights. Less meaningful detectors are also present either reflecting the higly over-parametrized nature of the RBM or a lack of learning. Given the good performance it is probable that the dataset could be sufficiently

represented by the other neurons.

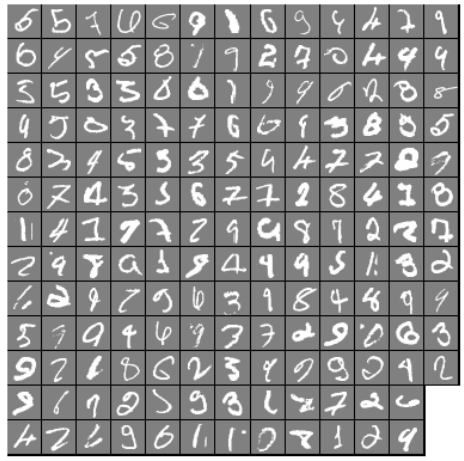

Figure 1.11: The 167 errors using a 3 layer DBN with 1000, 1000 and 1000 neurons respectively.

While some of the images are genuinely difficult to label, a number of them seems easy. Many of the sevens in particular seem fairly easy. The added intra-class variation due to continental sevens and regular sevens might explain this to a degree.

Hinton showed an error rate of 1.25% in his paper introducing the DBN and contrastive divergence [HOT06]. This impressive performance was achieve with a 3 layer DBN with 500, 500 and 2000 hidden units respectively, training a combined model of the representation and the labels in the last layer together and using extensive cross validation to tune the hyper parameters. Additionally Hinton used a novel up-down algorithm to tune the weights on the classification task, running a total of 359 epochs resulting in a learning time of about a week. It has been shown [VLL+10] that pre training a DBN, using its weights to initialize a deep FFNN and training that on a supervised task with stochastic backpropagation can lead to the same error rates as those reported by Hinton. As such it seems that it is not the training regime used resulting in the higher error rate but rather a need for further tuning of the hyper parameters.

1.3.2 Stacked Denoising Autoencoder

A three layer stacked denoising autoencoder (SDAE) with architecture identical to the DBN was created. The denoising autoencoder works just like the normal autoencoder except that the input is corrupted in some way, and the autoencoder is trained to reconstruct the un-corrupted input [VLBM08]. The idea is that the autoencoder cannot simply copy pixels and will have to learn corruption invariant features to reconstruct well. The corruption process used was setting a randomly selected fraction of the pixel in the input image to zero. The SDAE consisted of three denoising autoencoder (DAE) stacked on top of each other each with 1000 hidden neurons, and each trained in a greedy-layer wise fashion. Each DAE was pre-trained with a fixed learning rate of 0.01 and a batchsize of 10 for 30 epochs and with a corruption fraction of 0.25 i.e. a quarter of the pixels set to zero in the input images. The noise level was chosen based on conclusions from [VLL$^+$10]. Having trained the first DAE the training set was feed-forwarded through the DAE and the second DAE was trained on the hidden neuron states of the first DAE, and similarly for the third DAE. After pre-training in this manner the upwards weights and biases were used to initialize a FFNN with 10 output neurons in the same manner as for the DBN. The FFNN was trained with a fixed learning rate of 0.1, with a batchsize of 10 for 30 epochs. The performance was measured as for the DBN resulting in an error rate of 1.71% or 171 errors. The code ran for 41 hours.

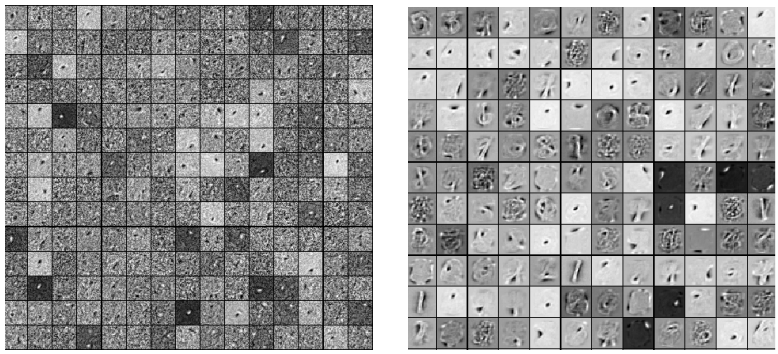

Figure 1.12: Left: Weights of a random subset of the 1000 neurons of a DAE with a corruption level of 0.25. Right: DAE trained in a similar manner with a corruption level of 0.5. The second DAE had worse discriminative performance. Its weights are shown here only to show that DAEs can find good representations. Each image is contrast normalized individually to be between minus one and one.

The DAE seems to find mostly seemingly non-sensical and blob detectors. For reference the weights of another DAE trained in a similar manner with a corruption level of 0.5 is provided. The latter finds stroke detectors and seem less noisy. The latter DAE had worse discriminative performance (not shown).

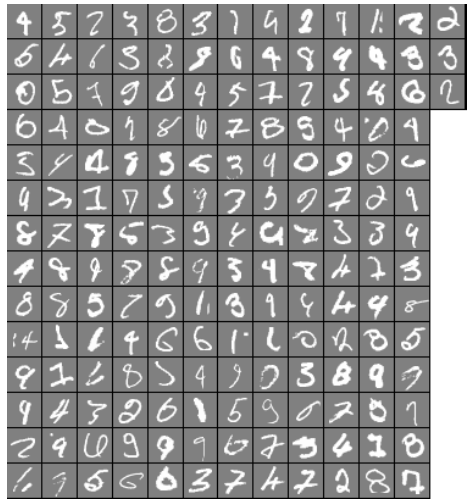

Figure 1.13: The 171 errors using a 3 layer SDAE with 1000,1000 and 1000 neurons respectively.

In [VLBM08] Pascal Vincent introduces the denoising autoencoder and reports superior performance on a number of MNIST like tasks. The basic MNIST test score is not reported though until his 2010 paper [VLL+10] in which Vincent reports an error rate of 1.28% on MNIST with a three layer SDAE using 25% corruption and extensive cross validation to tune the hyperparameters. The 1.71 % error rate here is on the same order of magnitude but compares unfavourably to the 1.28 %. It is evident that further tuning of the hyper parameters would have been beneficial.

1.3.3 Convolutional Neural Network

A Convolutional Neural Network was created following the architecture in [LCBD+90] in which Yann LeCun introduces the CNN. The first layer has 6 feature maps connected to the single input layer through 6 5x5 kernels. The second layer is a a 2x2 mean-pooling layer. The third layer has 12 feature maps which are all connected to all 6 mean-pooling layers below through 72 5x5 kernels. This full

connection between layer 2 and 3 is in contrast to the architecture proposed by LeCun, which used a hand-chosen set of connections. The fourth layer is a 2x2 mean-pooling layer. When training, the feature maps of this fourth layer is concatenated into a feature vector which feeds into the fifth and final layer which consists of 10 output neurons corresponding to the 10 class labels.
The CNN was trained with stochastic gradient descent on the full MNIST training set. A batch size of 50 and a fixed learning rate of 1 was used for 100 epochs resulting in a test score of 1.22% or 122 misclassifications. The code ran for 7 hours.

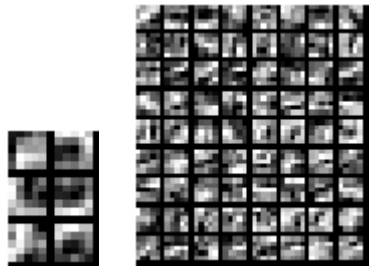

Figure 1.14: Left: The 6 kernels of the first layer. Right: The 72 kernels of the third layer. All kernels are contrast normalized individually to be between minus and plus one.

The CNNs 6 first layer kernels seems to be 4 curvy stroke detectors and two less well defined detectors. The 72 layer three kernels cannot be directly analysed with respect to what detectors they are as they operate on already transformed input. There does seem to be some structure in them though reflecting that the feature maps in layer 2 are still resembling digits.

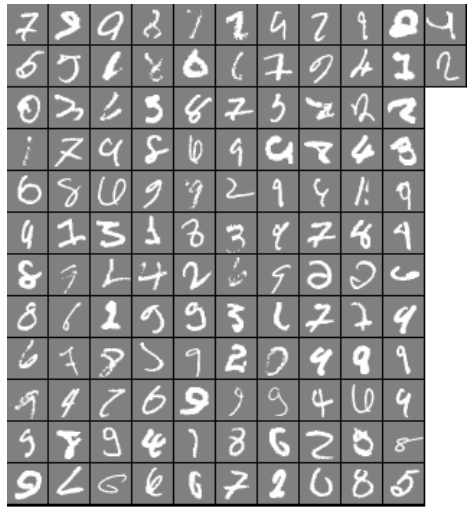

Figure 1.15: The 122 errors with the CNN.

The MNIST dataset did not exist at the time LeCun introduced the CNN. However in his 1998 paper [LBBH98] he reports a 1.7 % error for a network of this architecture, which he names LeNet1. Lecun subsamples the images to 16x16 pixels and uses a second order backprop method to achieve the 1.7%. The 1.22 % error rate compares favourably with this as there were no pre-processing and simple first-order backprop with a fixed learning rate was used. It should be noted that LeCun reports an error rate of 0.95% with LeNet-5, a more advanced net in the same paper. As no cross-validation was used to find the hyper-parameters the performance could probably be increased with further tuning of the hyper-parameters. It is remarkable that such a simple architecture as LeNet-1 is able to achieve such good performance.

Chapter 2

A prediction learning framework

2.1 Introduction

2.1.1 The case for a prediction based learning framework

As described autoencoders works by encoding a visible input to a hidden representation and decoding it back to a reconstruction of the input. The learning is then done by minimizing the difference between the reconstruction and the input. As described these models can just learn the identity and to achieve feature detectors akin to those expected, i.e. edge detectors or gabor-like filters, most authors see the need to make the reconstruction more challenging. Many papers use sparsity to this end, and describe that without this, the model did not find good feature detectors [LGRN09, GLS+09], others, such as the denoising autoencoder applies noise to the input image.

When reconstructing input, there are, obviously, no changes from the input to the output. In other words there are no transformations. It might seem curious then that we hope to find representations that are invariant to transformations, such as small translations, noise, illumination, etc. The logic is that if

we present the model with enough data where these transformations take place between samples, i.e. one sample may be a shifted or noisy version of another sample, then the model will learn representations invariant to the presented transformations. This makes sense as an invariant feature detector will on average create better reconstructions. If the model is over-saturated however, there is less need for sharing features amongst samples and heuristics are needed to achieve invariant feature detectors, as described above.

A more direct approach to achieve invariance might be to train the model on invariant output and transformed input; the transformations applied to the input being exactly the transformations we wish to achieve invariance to. Feature detectors would be forced to be invariant to something in order to reconstruct correctly and the sum of feature detectors should be invariant to all applied transformations.

The denoising autoencoder can be seen in this light as it adds noise to the input and reconstructs the clean input. The transformation applied here is noise and the invariance we achieve is invariance to noise. We could explore this further by rotating, translating, elastically deforming, etc. the input data and reconstructing the clean output. But adding the transformations we wish to achieve invariance to explicitly seems like a poor solution. Further, our human intuition about the desired invariances will not work as well when we add additional layers and need to describe the needed invariances of say, a rotation detector. Ideally we would like a model that could be trained fully unsupervised, and achieve invariances not limited to the extent of its authors understanding of the data.

I hypothesize that the transformations taking place over time are exactly the transformations we wish to achieve invariance to and as such, prediction would be a far better candidate for learning than reconstruction. In the case of video, the frame to frame differences include translation, rotation, noise, illumination changes, etc. In short all the transformations that we wish to achieve invariance to. A model predicting video frames would take some number of previous frames and attempt to predict the next frame. Training a model like this is an instance of the previously described more direct approach to learning in which each learning sample contains transformations.

Furthermore, prediction is a much harder task than simple reconstruction and it is hypothesized that as such the model would be much less prone to over fitting and see less need for heuristics such as sparsity, weight decay, etc.

From a biological standpoint prediction as a candidate for learning makes good sense. Being able to predict the environment and the consequences of ones actions is arguably what give intelligent life an advantage. Jeff Hawkins makes this argument at length in [HB04] and Karl Friston suggests minimization of free energy, or prediction error, as the foundation for a unified brain theory [Fri03]. One very appealing feature of prediction as a learning framework is that the learning signal, the prediction error, is unsupervised and readily available. There is no need for an external source of learning or complex setups for defining the learning signal. Instead predictions are made at all time at the neural level, and prediction error drives the learning. One of Jeff Hawkins ways to illustrate that the brain is making predictions at all times is to point out that you know what the last word in this sentence is before it ends.

Recent understandings [SMA00] of the mechanisms behind long term potentation and de-potentation in the neural connections can be seen in the light of the prediction learning framework as well. Specifically the theory of spike-timing dependent plasticity (STDP) describes that if the pre-synaptic neuron delivers input to the post-synaptic neuron shortly before the post-synaptic neuron fires, their connection is strengthened and if it delivers it after the post-synaptic neuron have fired, their connection is weakened. In short, if a pre-synaptic neuron can predict the firing of the post-synaptic neuron their connection is strengthened and if not, their connection is weakened. Experiments have shown that implementing a STDP learning rule in a network of artificial neurons can lead to the network predicting input sequences [RS01].

Prediction as a learning framework fits well with the array of evidence pointing to a vast amount of top-down connections in the brain[EGHP98], which are even expected to exceed the number of bottom-up connections [SB95]. It has been shown that these top-down connections modulate the bottom-up input at multiple stages in the visual pathway. [AB03, SKS$^+$05]. Also, there is an asymmetry in the bottom-up and the top-down connections effect. *"In particular, while bottom-up projections are driving inputs (i.e., they always elicit a response from target regions), top-down inputs are more often modulatory (i.e., they can exert subtler influence on the response properties in target areas), although they can also be driving"* [KGB07].

2.1.2 Temporal Coherence

Temporal coherence is tightly linked to prediction and as such will briefly be reviewed here. Temporal Coherence is the observation that, generally, the *objects*

giving rise to the sensory inputs are changing slowly over time. The *objects* in mention here could be trees, or the sun or concepts such as time of day, which are combined in a highly non-linear way to create the sensory inputs we perceive. Whereas the objects are slowly and smoothly changing, i.e. a person walking from one end of the room to the other, the sensory signals might be rapidly and seemingly randomly changing, i.e. the pixels seen might alternate quickly between black and white as the persons clothing folds and shadows are cast.

The principle of Temporal Coherence been attempted to be used for learning in the past [F̈91, Sto96] and more recently have been attempted as a learning principle for deep learning approaches. Two such approaches will be outlined shortly here to illustrate how the idea can be applied.

One application of temporal coherence is to force deep representation to be temporally coherent in some standard deep learning method as seen in [MMC+09]. The paper uses a convolutional neural net on a supervised learning task. It introduces an extra unsupervised learning signal though, such that their algorithm becomes:

- input a labelled image, and take a gradient step to minimize classification error.
- input two consecutive image from unlabelled video and take a gradient step to maximize the temporal coherence in layer N.
- input two non-consecutive images from unlabelled video and take a gradient step to minimize the temporal coherence in layer N.

This learns the model to classify labelled images while it forces the representations in layer N to be temporally coherent. The paper sets layer N as their next-deepest layer, the logic being that this should be the most high-level representation of the image and thus should be varying the slowest.

Another more direct application of temporal coherence is seen in Slow Feature Analysis (SFA). SFA focuses on extracting slowly varying time signals from time sequences. In short SFA seeks to transform a N dimensional time signal to an MN dimensional feature space in which each features temporal variance is minimized under the constraint that the features cannot be trivial i.e. being zero or having zero variance and that each feature should be different (uncorrelated)

[BW05]. How to find the MN dimensional feature spaces, e.g. learn in the model is explained in [WS02] and is beyond the scope of this thesis. It should be noted that SFA can be applied recursively, e.g. in a layer wise fashion where the output of one SFA is the input to the next.

SFA has been applied to natural images undergoing various transformations over time such as rotation, translation, etc, and lead to a rich repertoire of complex cell like filters [BW05] as well as successfully applied to artificial data [WS02].

2.1.3 Evaluating Performance

When designing deep learning modules, it is difficult to evaluate their performance quantitatively. Ultimately such modules are to be used in a deep architecture, on some classification task, but until the module itself have been shown to find good representations of the data there is little point in building a bigger model comprising many such modules to evaluate it using a classification task. As such the majority of the evaluation of the following proposed models will be qualitative, looking at what representations the modules find. As the proposed models all work on natural video the receptive fields found will be compared to the receptive fields of the primate visual cortex.

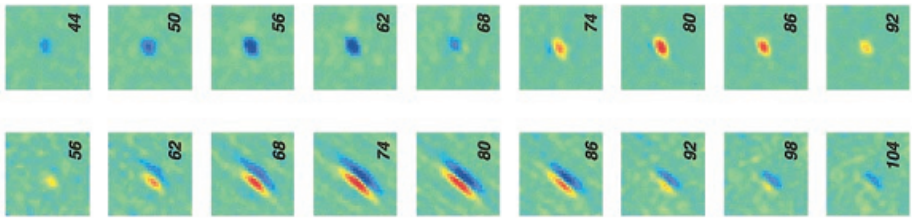

Figure 2.1: Estimated Spatio-Temporal receptive field of Macaque monkey. Top: Example of a receptive field resembling a blob-detector. Bottom: Example of an oriented spatio-temporal receptive field. Time in miliseconds is superimposed (upper right corner, tilted) on each image. Taken from [Rin02].

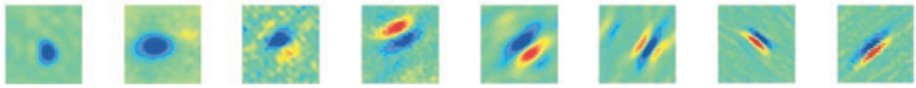

Figure 2.2: Estimated receptive field of Macaque monkey at their optimal time-delay. Several receptive fields resembling gabor filters at various orientations are visible. Taken from [Rin02].

Figure 2.1 and 2.2 shows the spatio-temporal and spatial receptive fields found in Macaque monkeys[Rin02] consistent with previous findings [HW68]. For models trained on video of natural scenes these are the kind of receptive fields the modules are supposed to find, i.e. oriented gabor filters.

2.2 The Predictive Encoder

The following section introduces the Predictive Encoder (PE) as a candidate for a prediction based deep learning model.

2.2.1 Method

The Predictive Encoder (PE) is an autoencoder that instead of reconstructing an input, attempts to predict future input, given n previous inputs. It encodes the concatenated previous inputs into a hidden representation and decodes the hidden representation to a prediction of the future inputs.

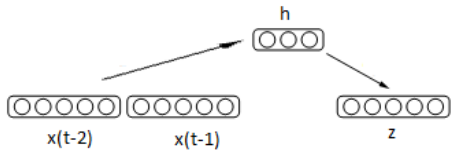

Figure 2.3: Predictive Encoder with $n = 2$ previous input. Modified from [Ben09].

The governing equations of the PE are very similar to that of a simple autoencoder.

$$h(x) = f(W_1 x_{t-1,t-2,\ldots,t-n} + b) \tag{2.1}$$
$$z(x) = f(W_2 h + c) \tag{2.2}$$
$$L(x) = \frac{1}{2}||z - x_t||^2 \tag{2.3}$$

Where h is a vector of hidden representations, f is a non-linearity, typically sigm or tanh, W_1 is the encoding weight matrix, $x_{t-1,t-2,\ldots,t-n}$ is a concatenated vector of the n previous inputs, b is a vector of additive biases for the hidden representations, z is a vector representing the prediction given by the PE, W_2 is the decoding weight matrix, c is a vector of additive biases, L is the sum square loss function and x_t is the input at time t, which the PE is trying to predict. Learning is done by updating the parameters W_1, W_2, b, c with stochastic gradient descent.

The PE can be trained layer-wise and stacked on top of each other, each trained on the hidden states of the PE below creating the Stacked Predictive Encoder (SPE) comparable to stacked auto-encoders or DBNs.

2.2.1.1 Relation to other models

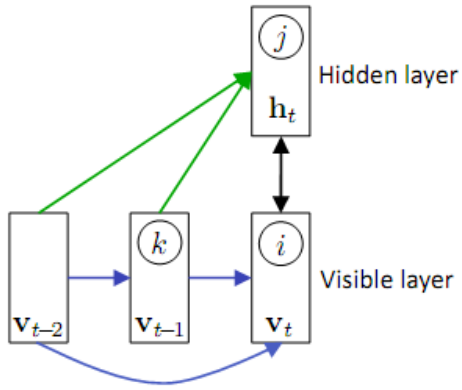

Figure 2.4: CRBM with $n = 2$ previous input. Taken from [THR11].

The Predictive Encoder bears close resemblance to a Conditional Restricted Boltzmann Machine [THR11] (CRBM). The main differences are that the CRBM

has directed connections between the previous inputs and the input to be predicted, undirected connections between the hidden layer and the input to be predicted, has stochastic neurons and uses contrastive divergence instead of backprop for learning. There is no reason the Predictive Encoder cannot have connections from previous inputs directly to the input to be predicted as the CRBM does. In the CRBM these connections primarily serve to model simple spatio-temporal pixel to pixel correlations, in effect a kind of whitening [THR11] which frees up the hidden layer to focus on higher order dependencies. If the data has been pre-whitened these might be less useful. CRBMs have been used in modelling motion capture data [THR11] and phone recognition [MH10] with good results for both. The predictive encoder can be seen as a simpler deterministic variant of the CRBM, which is simpler to train.

Two related and highly influential models of cortical functions are Predictive Coding (PC) [SLD82, RB99] and Biased Competition (BC) [DD95, Bun90]. Predictive coding is a model of cortical function hypothesizing that top down information predicts bottom up information, and inhibits all bottom up information consistent with the prediction. In this way only the error signal is propagated upwards, which allows for efficient coding of redundant/predictable structures. Biased Competition is the seemingly opposite theory that top-down information enhances bottom-up information consistent with the top-down information, serving to solve a competition for activity, i.e. solve which neural representation gets to represent the input. The two models have been shown to be equal under certain conditions [Spr08]. PC has been shown to replicate end-stopping and other extra-classical receptive field effects [RB99] and two PC models have been shown to lead to gabor like receptive fields when trained on natural images [RB99, Spr12]. The proposed predictive encoder differs from these models in that while they both mention an extension to temporal data, they concentrate on the spatial predictions whereas the predictive encoder is only defined for temporal data and inherently learns on spatio-temporal patterns.

The predictive encoder can be seen as a denoising autoencoder which applies a corruption process learned naturally from the data. Vincent ends his thorough paper on denoising autoencoders discussing the benefits of this. *"If more involved corruption processes than those explored here prove beneficial, it would be most useful if they could be parametrized and learnt directly from the data, rather than having to be hand-engineered based on prior-knowledge."* [VLL+10]. As mentioned it is hypothesized that the variations occurring over time are exactly equal to the noise we wish to achieve invariance to and as such should prove beneficial. The PE and the DAE differ in that the former is defined for temporal data whereas the latter is defined for non-temporal data. The PE re-

duces to a denoising autoencoder if the only noise occurring over time is random and independent, such as a noisy neuron looking at the same input over several time-steps.

2.2.2 Dataset

The dataset proposed for measuring invariances [GLS+09] was used. It consists of 34 videos of natural scenes sampled at 60 frames per second at a spatial resolution of 640x360. In total there are 11.116 frames corresponding to roughly 3 minutes of video. The dataset has been whitened by applying a pass-band filter and contrast normalized with a *"scaling constant that varies smoothly over time and attempts to make each image use as much of the dynamic range of the image format as possible"* [GLS+09]. The dataset contains a number of common variations such as translation, rotation, differences in lighting, as well as more complex variations such as animals moving or leaves blowing in the wind. Segmenting or classifying natural video is a particular challenging machine learning task. Unlike hand-written digits the underlying distribution giving rise to natural video is not low-dimensional, indeed its underlying distribution probably has a higher dimension than the pixel representation. This specific set of natural video is particular suitable due to the high frame rate at which it was captured, and the good quality of the video.

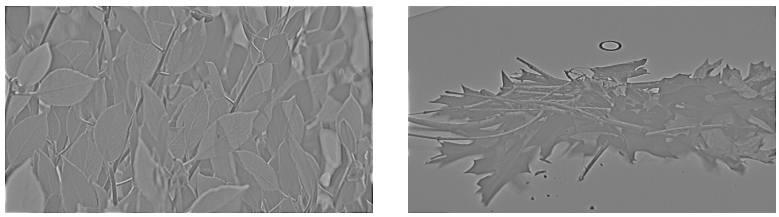

Figure 2.5: Two random frames from the dataset used.

2.2.3 Results

The images comprising the dataset was split into patches of size 10x10 with no overlap, resulting in 2304 patches per image, or roughly 25 million patches in total from which a random subset of 100.000 patches were chosen.
A predictive encoder and an auto-encoder was created to model the patches. Both the PE and the AE had 100 hidden units, used a small amount of L2

weight decay of 0.00001, had a fixed learning rate of 0.1, a batch size of 1 and was trained for 100 epochs resulting in ten million gradient descent steps. The predictive encoder had $n = 2$, i.e. was given the two previous patches in the same position and was trained to predict the given patch. The auto-encoder was given the patch and trained to reconstruct the patch. The code ran for approximately 6 hours.

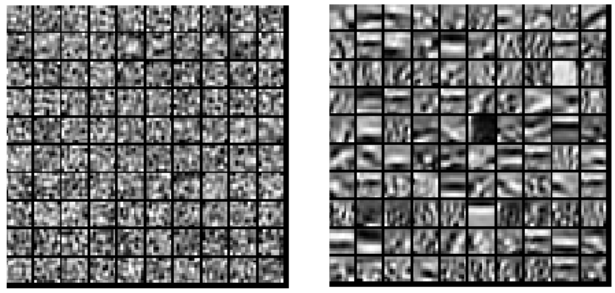

Figure 2.6: Left: 100 output weights of an auto-encoder trained on natural image patches. Right: 100 output weights of a predictive encoder trained on natural image patches. Each image is contrast normalized to between minus one and one.

As expected the auto-encoder did not learn a useful representation as the model was not subjected to any sparsity constraints. The PE however did learn a very interesting representation; amongst the receptive fields are oriented line detectors, oriented gratings, gabor-like filters, a single DC filter and more complex structured filters.

To compare to other methods two denoising autoencoders were trained on the same data, again for 100 epochs, a fixed learning rate of 0.1 and a batch size of 1. Two levels of zero-masking noise levels was tried for the DAEs, namely 0.5 and 0.25, which both gave the same results.

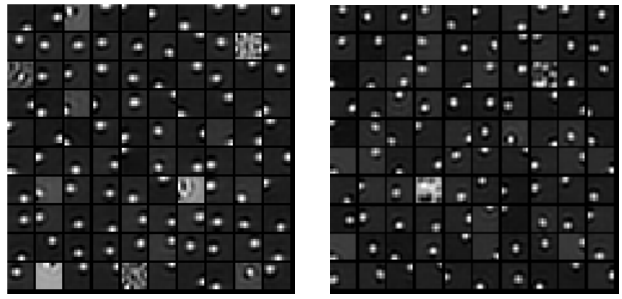

Figure 2.7: 100 output weights of a denoising auto-encoder trained on natural image patches Left: zero masking fraction of 0.5. Right: Zero masking fraction of 0.25. Each image is contrast normalized to between minus one and one.

It is evident that for this data and training regiment, the denoising autoencoder did not find interesting features. It should be noted that denoising autoencoders have been shown to find gabor like edge detectors and oriented gratings when trained on natural image patches [VLL+10]. In that study the author used a linear decoder, tied weights and various corruption processes each leading to slightly different results. There seems to be no reported attempts at using standard RBMs for modelling natural image patches, but more advanced RBM-like models, including spike-and-slab RBMs [CBB11] and factored 3-way RBMs [THR11] have been used successfully leading to gabor-like filters.

It is interesting to examine the levels of sparsity of the models trained as we have a strong basis of evidence for sparse coding in the brain. To do this all 100.000 patches were fed through the models, the activations of their hidden units were recorded and a histogram of all hidden neuron activities for all patches were created for each model.

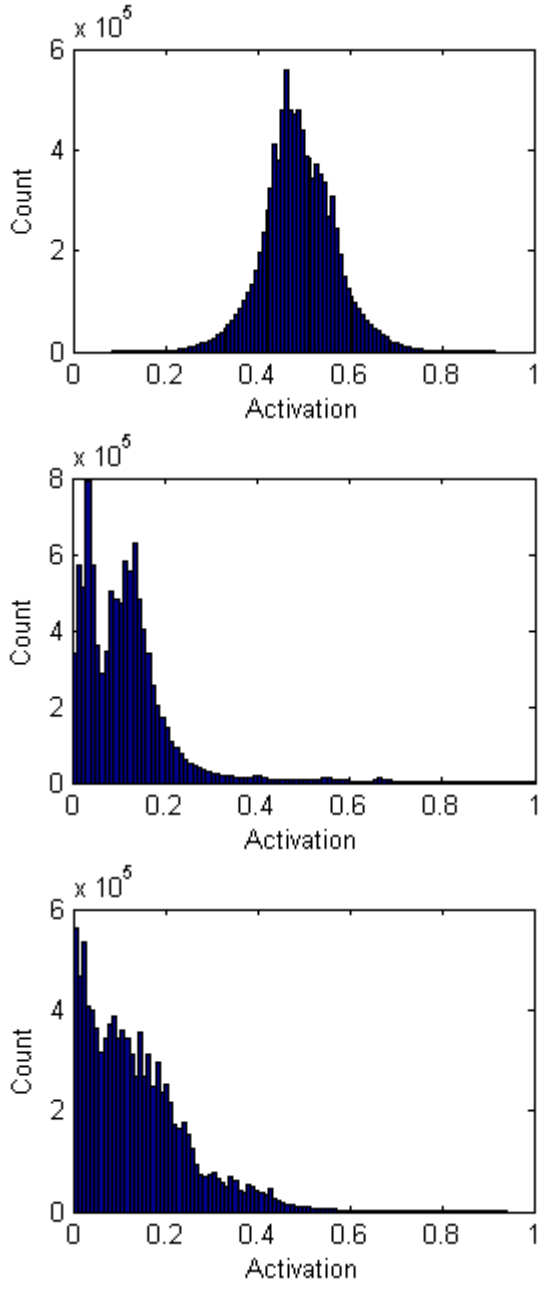

Figure 2.8: Hidden neuron activations for natural image patches. Top: AE. Middle: PE. Bottom: DAE.

Figure 2.8 shows that the AE did not find a sparse representation while the PE did find a sparse representation, which seems to be bi-modal with increased counts at around 0.4, 0.6 and 0.7 levels of activation. The DAE finds a slightly less sparse representation which also seems slightly bi-modal, but without the increased counts at higher activation level. It is interesting to note that the trivial blob detectors found by the DAE leads to a somewhat sparse representation. As such, a sparsity constraint does not seem to be a guarantee for finding a good representation.

The PE is different from the other models in that it is looking at spatio-temporal patterns. To visualize what the PE is picking up we can look at the input weights partitioned into those looking at the patches at different time steps.

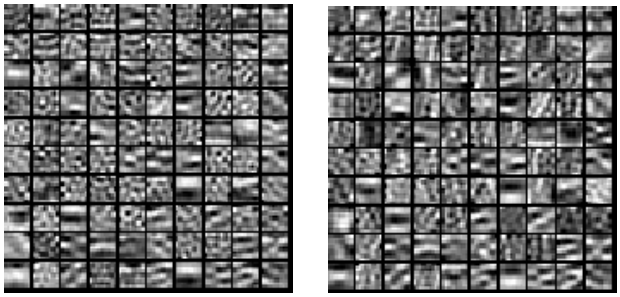

Figure 2.9: Left: first 100 input weights of a PE trained on natural image patches. Left: second 100 input weights of a PE trained on natural image patches. Each image is contrast normalized to between minus one and one.

Most of the neurons are nearly equal in the two images, reflecting the high degree of frame to frame similarity, but a few neurons are picking up temporal patterns, which is easy to see when rapidly flickering between the two images, less so when viewed on paper. If the image grid is a standard coordinate system then neuron (3,8) can be seen as a rotation detector as it picks up on lines in one direction in the first frame and a lines in the perpendicular direction in the second frame. Neuron (7,1) and (7,4) are curiously picking up on the same input, which are vertically moving lines.

It has been shown that predicting the frame-to-frame variations in natural image patches with a very simple model lead to a sparse hidden representation with receptive fields similar to those found in the visual cortex. These results are

highly encouraging for prediction learning as a framework. Models used on natural images are usually much more complex, as in the case of the factored RBM, employs sparsity as an explicit optimization goal or uses hand-crafted architecture and or hyper-parameters as in the case of CNNs and DAE. While the feature detectors found were not perfect, it is likely that much better feature detectors could be found with a more thorough search of the hyper-parameters space and by examining similar models, e.g. a PE with a linear decoder, or with visible to visible connections as in the conditional RBM.

2.3 The Convolutional Predictive Encoder

The following section introduces the Convolutional Predictive Enoder (CPE) as a candidate for a prediction based deep learning module that scales to realistic size images.

2.3.1 Method

The CPE is a natural extension of the PE built to scale the PE up from patches to realistic size images. A convolutive model was decided upon for computational speed and built-in translation invariance. The Convolutional Predictive Encoder is very similar to a convolutional autoencoder [MMCS11], but instead of encoding/decoding the input to itself it encode/decodes the input into future input and instead of working on images, it works on stacks of images. In the following the stacks of images are oriented such that the first dimension is time, and the second and third dimension are spatial, i.e. $(2, 3, 4)$ refers to pixel (3,4) in image 2.

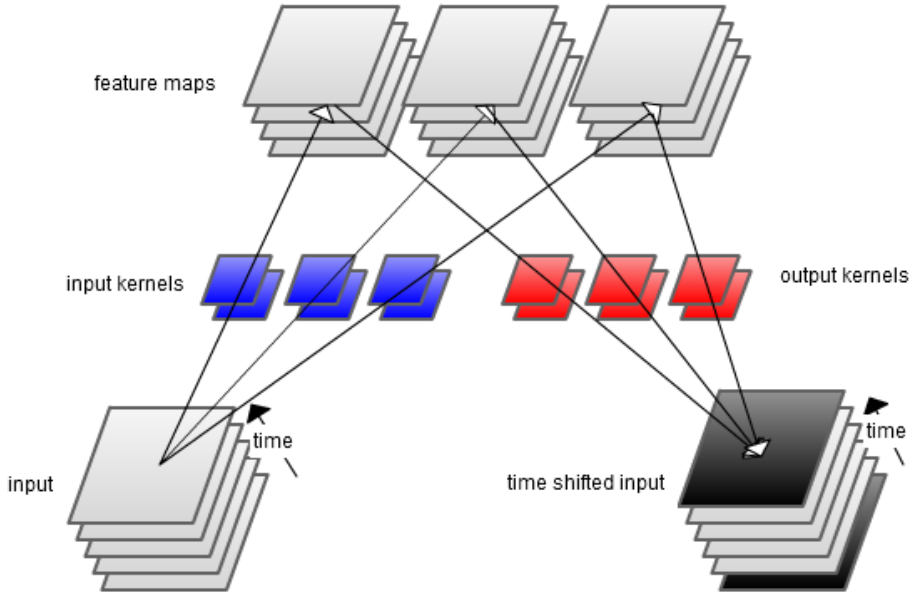

Figure 2.10: Convolutional Predictive Encoder one input cube, 3 feature cubes, input kernels and output kernels and one output cube. The black images in the output cube are not used for learning due to edge issues as described in the text.

The CPE encoding works by convolving an input cube, with a number of input kernels, adding a bias and passing the result through a non-linearity resulting in a number of feature cubes. The activations A_j of a single feature cube j is

$$A_j = f\left(b_j + \sum_{i \in M_j} I_i * ik_{ij}\right) \tag{2.4}$$

Where f is a non-linearity, typically tanh() or sigm(), b_j is a scalar bias, M_j is a vector of indexes of the input cubes I_i which feature cube j should sum over, $*$ is the three dimensional convolution operator and ik_{ij} is the kernel used on input cube I_i to produce input to the sum in feature cube j. In figure 2.10 there is only a single input cube and as such the sum is redundant in this case. The sum has been included to allow multiple input cubes to be present, e.g. stereo vision. Following [MMCS11] a max pooling operation was included in the feature cube layer. This max pooling operation sets all values of the feature cube to zero except for the maximum value, which is left untouched, in non-overlapping cubes of size (s_x, s_y, s_z). This effectively forces the feature cube representation to be spatio-temporally sparse with a fixed activation fraction

of $1/(s_x s_y s_z)$. This sparsity constraint might not be necessary to use when working in prediction mode, as the prediction problem is much harder than reconstruction, but it was included to allow comparison to [MMCS11] and to give good results in reconstructive mode.

$$A_j = \text{maxpool}_{s_x, s_y, s_z}(A_j) \tag{2.5}$$

Similarly, the decoder works by convolving the feature cubes with output kernels, summing all the results, adding a bias and passing it through a non-linearity. The decoded output O_i is

$$O_i = f\left(c_i + \sum_{j \in N_i} A_j * ok_{ij}\right) \tag{2.6}$$

Where c_i is a scalar additive bias, M_j is a vector over which feature cube should contribute to this output cube and ok_{ij} is the output kernel associated with output cube i and feature cube j. In figure 2.10 $M_{1,2,3} = [1]$ and $N_1 = [1, 2, 3]$, i.e. the CPE is fully connected.

The loss function is given as the sum squared difference between each output cube O_i and the time-shifted input cubes $\hat{I}_i$.

$$L = \frac{1}{2} \sum_i ||O_i - \hat{I}_i||^2 \tag{2.7}$$

The time shifted input cubes are the input cubes shifted N time step forward, where N is the depth of the input and output kernel. If the input cubes are stacked video frames then the bottom frame corresponds to the first frame, i.e. $t = 0$. In 2.10 the kernel depth is 2 and as such in the time shifted cube the bottom frame would then the third frame, i.e. $t = 2$.

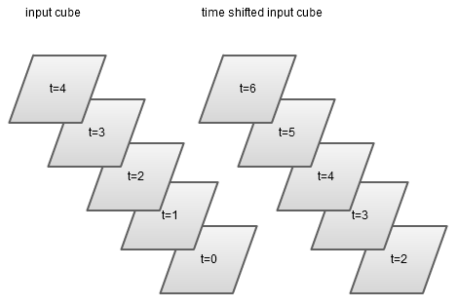

Figure 2.11: Left: stack of input frames. Right: input frames time-shifted 2 steps.

This ensures that the output frame at $t = T$ only receives input from input frames $t < T$ and as such is cannot reconstruct but has to predict. More precisely if the kernel is N deep in the time-dimension (the kernels in figure 2.10 are 2 deep), the output frame at $t = T$ receives input from input frames $T - 2N \leq t \leq T - 1$.

The model is trained with stochastic gradient descent to minimize the loss function.

$$\Delta\theta = \alpha \frac{\partial L^{(i)}}{\partial \theta} \tag{2.8}$$

$$\theta = \theta - \Delta\theta \tag{2.9}$$

Where θ is a parameter, i.e. a bias or kernel weight, α is the learning rate, and $L^{(i)}$ is the instantaneous loss when given input cube i. A derivation of the partial derivatives of each parameter in one dimension can be found in appendix. The derivations for n-dimensions are the same, just with n dimensions. In short, computing the gradients can be done in a backprop-like way where the error is propagated backwards in the network using convolutions.

When using the convolution operator there is a problem of edges. Convolving two 1 dimensional signals of length n and m gives a resulting signal of length $n - m + 1$. The same holds true for n-dimensional signals. After two convolutions corresponding to encoding with an n_e size kernel and decoding with a n_d size kernel, the resulting signal is $n - m_e - n_d + 2$. In other words the output cube is smaller than the input cube. To overcome this problem the input cube is padded with zeros. This results in equal sized input and output cubes, but the output cube is not correct as the edges of the output cube receive input

from the zero padded region of the input effectively receiving less input than the central parts. If the model is trained in this way the kernels will learn poor representations as they collapse in the edges and corners. To overcome this the error is multiplied element-wise with a mask of zeros such that the errors at the edges become zero and thus do not contribute to the learning. Specifically, if the input/output cubes are size (x, y, z) and the input and output kernels are size (k_x, k_y, k_z) then the input cube is padded with $(k_x, k_y, k_z) - 1$ zeros and the $(k_x, k_y, k_z) - 1$ outermost parts of the output cube is zero masked. In figure 2.10 the black images, i.e. the first and last images are zero masked. Not shown is the border of the inner images which are also zero masked. Other more advanced methods for handling edges have been shown to give good results as well [Kri10].

To speed the learning a second order methods is used. The idea in second order methods is to approximate the loss as a function of the parameters locally with a second order polynomial. This is a better approximation than the first order polynomial employed with gradient descent as it takes into account the curvature of the loss function. This avoids taking too large steps when the curvature is high, and increases the step size when the curvature is small. However, the second order methods are not well defined for stochastic learning [LBOM98] and instead a stochastic diagonal levenberg-marquardt method [LBOM98] is used. In it each parameter θ_i is given its own learning rate η_i.

$$\eta_i = \frac{\alpha}{\mu + \frac{\partial^2 L}{\partial \theta_i^2}} \tag{2.10}$$

Where $0 < \mu < 1$ is a safety factor added to keep the learning rate from blowing up when the second derivatives goes to zero. The second derivative of the sum squared loss function with respect to all parameters θ is

$$L = \frac{1}{2} \sum_{n=1}^{N} (O^{(n)} - \hat{I^{(n)}})^2 \tag{2.11}$$

$$\frac{\partial L}{\partial \theta} = \sum_{n=1}^{N} (O^{(n)} - \hat{I^{(n)}}) \frac{\partial O^{(n)}}{\partial \theta} \tag{2.12}$$

$$\frac{\partial^2 L}{\theta \theta^T} = \sum_{n=1}^{N} \frac{\partial O^{(n)}}{\partial \theta} \frac{\partial O^{(n)}}{\partial \theta}^T + (O^{(n)} - \hat{I^{(n)}}) \frac{\partial^2 O^{(n)}}{\partial \theta \partial \theta^T} \tag{2.13}$$

The second order partial derivatives are dropped as well as the off diagonal terms as an approximation and a way to ensure that the learning rate stays

positive [LBOM98].

$$\frac{\partial^2 L}{\partial\theta\partial\theta^T} \approx \sum_{n=1}^{N} \frac{\partial O^{(n)}}{\partial\theta} \frac{\partial O^{(n)}{}^T}{\partial\theta} \tag{2.14}$$

$$\frac{\partial^2 L}{\partial\theta_i^2} \approx \sum_{n=1}^{N} \frac{\partial O^{(n)}{}^2}{\partial\theta_i} \tag{2.15}$$

A derivation of this approximation can be found in appendix. The second order derivatives can be evaluated on a subset of the training data and only needs to be re-evaluated every few parameter updates due to the slowly changing nature of the second derivatives [LBOM98].

To further decrease learning time momentum is used [YCC93]. Momentum is a trick to increase the learning rate in long narrow ravines of the loss function and decrease oscillations along the steep edges of the ravine. Instead of applying the calculated parameter update directly, it is used to change the velocity of the parameter update.

$$v = \beta v + \Delta\theta \tag{2.16}$$

$$\theta = \theta - v \tag{2.17}$$

Where β is a constant between 0 and 1 that specifies how much momentum the parameter updates should have, i.e how much they should remember earlier parameter updates.

2.3.1.1 Relation to other models

There are surprisingly few attempts at learning invariant feature detectors from video, perhaps attributed to the computational cost of working with video. State of the art approaches instead uses hand coded spatio-temporal interest points [Lap05, KM08], which, as they are not learned, are outside the scope of this thesis. The two most prominent deep learning methods attempting to learn these feature detectors will be briefly introdued.

The model most similar to the CPE is the Space-Time Deep Belief Network (ST-DBN) [Che10]. The ST-DBN model is built of alternating layers of spatial and temporal convolution each followed by probabilistic max pooling as introduced by [LGRN09] using convolutional RBMs (CRBM) as the learning module.

The CRBM learns using a hybrid of contrastive divergence and stochastic gradient descent to maximize the log likelihood of the training data while explicitly enforcing a sparsity constraint on the max-pooled units.

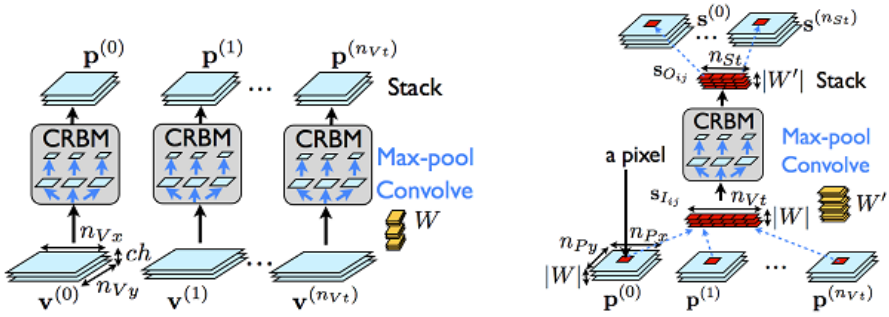

Figure 2.12: Space-Time DBN (ST-DBN). Spatial layer - Left: A sequence of images $v^{(0)}, v^{(1)}...v^{(n_{Vt})}$, each arranged as a stack of three color channels, are each convolved with a number of spatial kernels W into $|W| * n_{Vt}$ feature maps, which are then max pooled into $p^{(0)}, p^{(1)}...p^{(n_{Vt})}$ pooled spatial feature maps. Temporal layer - Right: The temporal layer takes the pooled spatial feature maps as input, concatenates all pixels at identical positions and sequential times into a long matrix representing 'feature activity' at 'time'. This matrix is then convolved, with a number of temporal kernels W', max-pooled in the time dimension, and re-arranged back into the original two-dimensional pattern of a video frame. Taken from [Che10].

The ST-DBN was trained on the invariance measure dataset [GLS+09], and showed increased invariance over a convolutional DBN, was applied to the KTH human action dataset [SLC04] and showed competitive performance and was applied to video denoising and unmasking with some success. The ST-DBN differs from the CPE in that the ST-DBN divides the spatial and temporal convolution and max-pooling into two seperate layers. Also, the ST-DBN is trained to reconstruct, rather than predict, and thus is highly overparameterized leading to the need for a sparsity constraint. In contrast the CPE extracts spatio-temporal features in a single layer and uses prediction as its learning signal.

Taylor et al. proposed a multi-stage model to learn unsupervised features from video for recognizing human activities in the KTH and Hollywood2 dataset [TB10]. The model first divides the video into local space-time cubes of a manageable size and uses Gated RBMs (GRBM) [MH07] to model the frame-

to-frame transformations. In a gated RBM, the hidden features, or so called 'transformation-codes' modulates visible frame-to-frame weights, thus allowing the features to modulate lower level correlation between successive frames. The model then learns a sparse dictionary of the latent transformation-codes, which finally, are max-pooled and fed into a SVM for classification. The model achieves competetive results on both the KTH dataset and the Hollywood2 dataset. The CPE, in contrast is a simpler architecture and only uses a single building block.

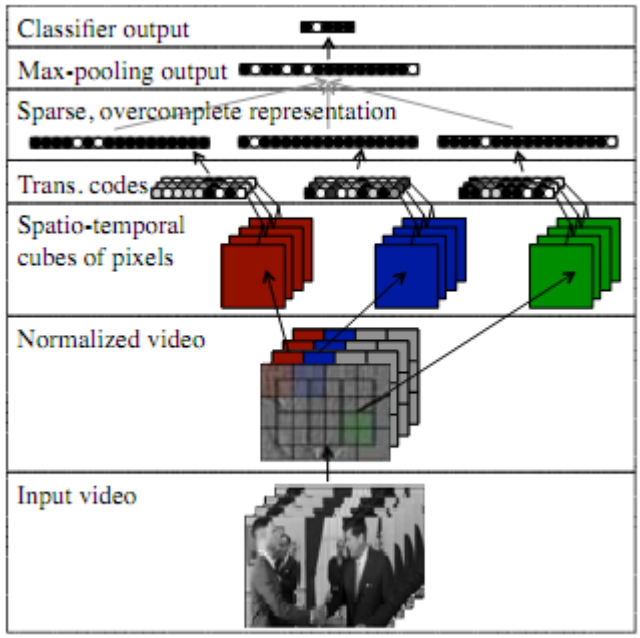

Figure 2.13: The model proposed by Taylor et al. Taken from [TB10].

2.3.2 Results

In the following experiments the same dataset as for the predictive encoder was used. This time the images were not made into patches but were resized to 160x90 for computational efficiency.

The first experiment was a convolutional predictive encoder with 20 input/output kernels (i.e. 20 feature cubes) of size (2x5x5). Conservative hyperparameters were chosen: a fixed learning rate of 0.001, μ of 1 such that the second order

method could not speed up learning in in flat regions of parameter space only slow it down in curved regions, momentum of 0.5 and a max pooling scale of $(1, 1, 1)$ equal to no max pooling. A batchsize of 10 was chosen, and the second order gradients were re-calculated for every parameter update. The code was run for 10.000 parameter updates, giving $10.000 * 10/11.116 \approx 9$ epochs which took around 14 hours. The results were not encouraging.

Figure 2.14: Top map of 20 output kernels of a CPE without max-pooling. All images are contrast normalized to between minus one and one.

Figure 2.14 shows the top layer of all output kernels of the trained CPE, which can best be described as noise. The bottom layer was equally noisy. The noise arises because the images are all very close to zero but contrast normalized to be between -1 and 1. Several values of hyper-parameters were tried for multiple runs of similar models. None of them gave better results.

To assert whether the implementation was faulty, experiments like the ones on MNIST in [MMCS11] were created to enable comparison. The experiments described in [MMCS11] uses Convolutional Auto-Encoders (CAE) which are exactly equal to the PCE, if one does not time-shift the output cube when calculating loss, i.e. they reconstruct, instead of predicting. CAEs are heavily over-parameterized, as each feature map contains about as much information as the input image. As such a CAE with a single feature map can reconstruct any image very well by simply copying the pixels. The non-linearities make this a bit more complicated, but the problem remains: a CAE with K feature maps are approximately K times over-complete [LGRN09]. Three CAEs were created and trained on the MNIST dataset. All CAE had 20 feature maps, 20 input/output kernels of size 5x5, $\mu = 1$, momentum = 0.5, a fixed learning rate of 0.001, batchsize of 10 and ran for 18.000 parameter updates or $18.000 * 10/60.000 = 3$ epochs. The code ran for approximately 1 hour. With the second order method and momentum they quickly converged, and after approximately 10.000 parameter updates the improvements in reconstruction were small. The second order gradients were re-calculated for each parameter update. The first CAE was as described, the second was a denoising CAE, by zero-masking 50% of the input image pixels and the third CAE was max-pooled with a pool-size of

2x2. In the following figures each image is contrast normalized individually to be between minus one and one.

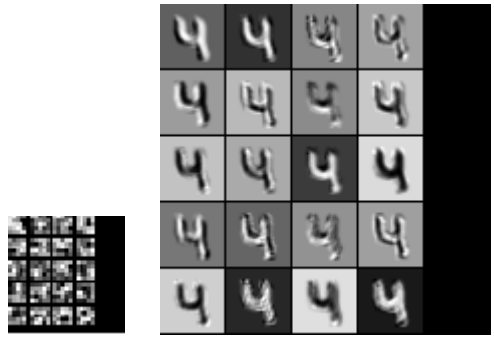

Figure 2.15: Left: Kernels of a CAE trained on MNIST. Right: The 20 feature maps for a single sample.

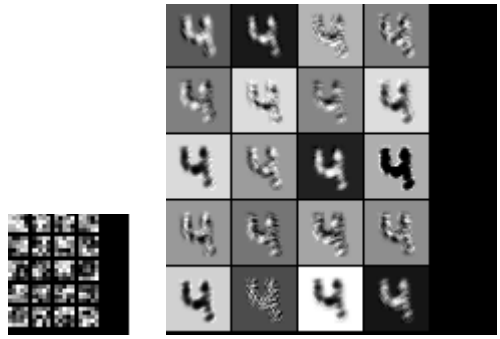

Figure 2.16: Left: Kernels of a denoising CAE trained on MNIST. Right: The 20 feature maps for a single sample.

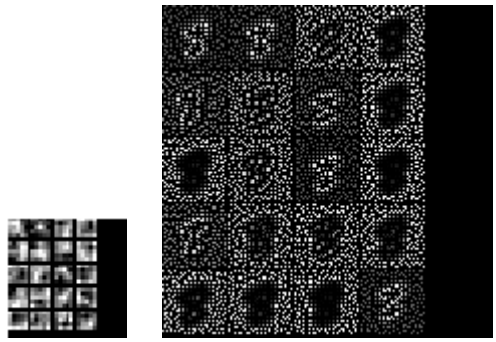

Figure 2.17: Left: Kernels of a max pooled (2x2 pool) CAE trained on MNIST. Right: The 20 feature maps for a single sample. The effect of max-pooling is visible.

The results are in accordance with [MMCS11] in that the kernels found for the CAE and DCAE are fairly noisy, while the max pooled CAE found better kernels. While the kernels found by the max-pooled CAE were better they were still not quite what one would expect, i.e. stroke detectors. In an effort to achieve better kernels a final CAE was trained. The CAE was identical to the ones described above, except its kernels were 9x9 and its pooling size was 9x9 thus heavily constraining the representational power of the feature maps.

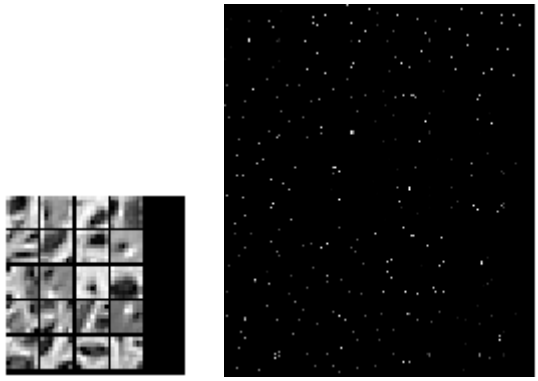

Figure 2.18: Left: Kernels of a max pooled (9x9 pool) CAE trained on MNIST. Right: The 20 feature maps for a single sample. The effect of the large max-pooling size is visible.

This finally gave good kernels resembling stroke detectors, gabor filters and big blob detectors. The effect of the max-pooling is visible in the very sparse representation in the feature maps.

Having shown that the CPE/CAE implementation can replicate findings and, with a sufficiently large pooling, size leads to good kernels on MNIST there are two possible reasons for the failure of the CPE. 1) the fault is in the the time-shifting part, i.e. when using the model for prediction rather than reconstruction or 2) the data used is faulty. A third possible explanation is that prediction is simply not a good learning signal for learning invariant features, but this would contradict the earlier findings.

To ascertain whether the fault was with the data or the time-shifting mode, a CAE, similar to the latest CAE trained on MNIST was created and trained in reconstruction mode on the the natural images. If this experiments succeeded in learning good features the fault would probably be in the time-shifting part of the model, i.e. when using it for prediction. The CAE was identical to previously described CAEs except it had 2x11x11 input/output kernel sizes and a max-pooling size of 1x10x10 as this parameter region had been shown to find the best kernels on the MNIST dataset. The CAE was trained with more aggressive hyperparameters of $\mu = 0.1$, momentum $= 0.9$ and a fixed learning rate of 0.01. The aggressive hyperparameter settings was found to be possible and beneficial in these heavily max-polled architectures. The results were not good.

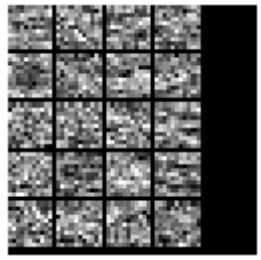

Figure 2.19: Weights of a CAE with 1x11x11 kernels and max-pool size of 1x10x10.

The kernels look random, which is because they are all very close to zero and contrast normalized. That the kernels are all zero are evident from the predictions the model made.

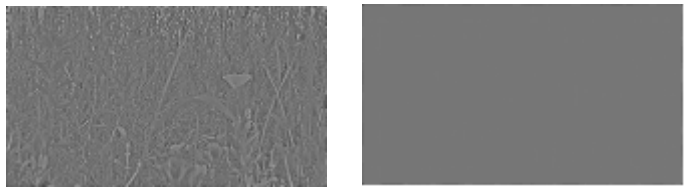

Figure 2.20: Left: Correct image. Right: Predicted image.

Seeing as the results were also poor in reconstruction mode, the error probably stems from the data used. The main difference between the data used in the PE and in the CPE is that the PE works on patches cut from the original un-resized data, whereas the CPE works on whole, resized, images. Looking closer at the resized data it is evident that much of what was structure in the original image, resembles noise after resizing. This is probably due to the combined effect of whitening followed by resizing. Whitening removes local pixel correlation while the resizing algorithm introduces it by taking means over local regions.

Having made apparent that it was the resizing employed that caused the problems a series of new experiments were run on the non-resized data. A central 190x160 patch was cut out of each image, giving the same data size as previously, but without the resizing problem. The series of experiments were run with spatial max-pooling sizes varying from one to four, and temporal max pooling varying between one and two. The experiments were all run using the conservative hyper-parameter setting, with 20.000 parameter updates. Each experiment took approximately 70 hours to run. The kernels found can be seen below.

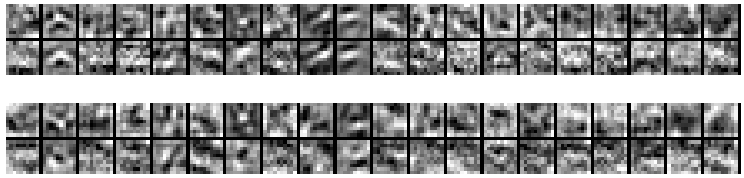

Figure 2.21: CPE kernels. No max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

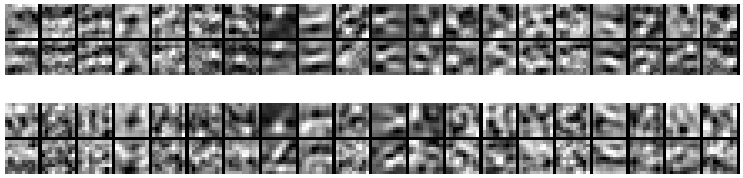

Figure 2.22: CPE kernels. 2x2x2 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

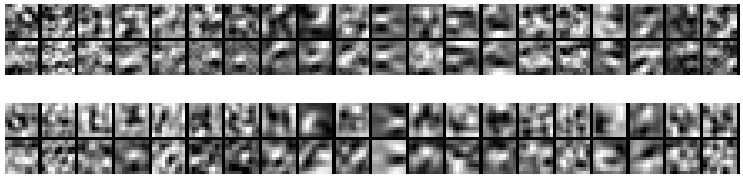

Figure 2.23: CPE kernels. 1x2x2 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

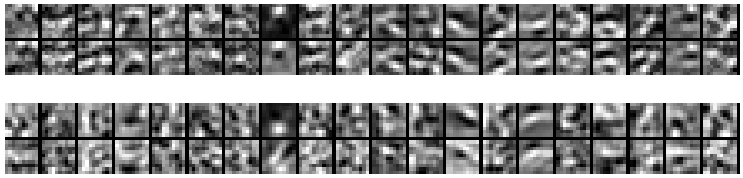

Figure 2.24: CPE kernels. 2x3x3 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

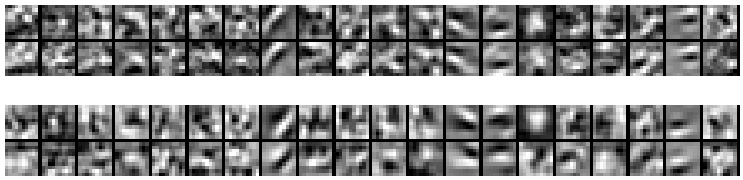

Figure 2.25: CPE kernels. 1x3x3 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

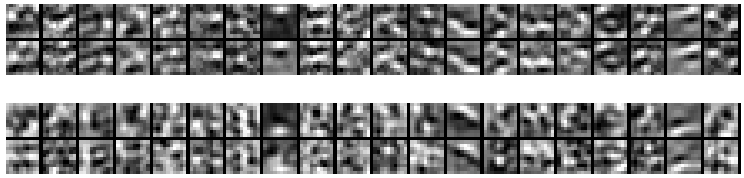

Figure 2.26: CPE kernels. 2x4x4 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

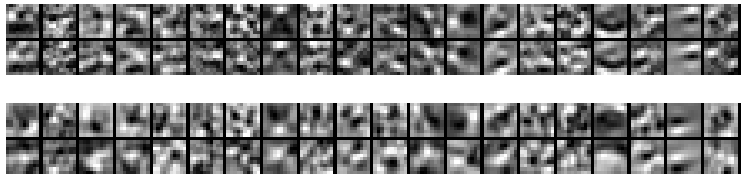

Figure 2.27: CPE kernels. 1x4x4 max-pooling. Top: Input kernels. Bottom: output kernels. Time goes from top to bottom.

All of the experiments has kernels showing structure, even the first experiment without max-pooling, although all experiments also feature seemingly noisy kernels. The first experiment without any max-pooling finds two gabor-like filters, a corner detector, several blob detectors and several noisy kernels. Interestingly there is a difference in the input and output kernels; the input kernels are mostly the same over time, whereas the output kernels changes over time. This phenomena can be seen in figure 2.25 the most clearly. It is evident that introducing spatial max-pooling results in better kernels, while temporal max-pooling seems to result in slightly worse kernels. The CPE with 1x3x3 max pooling seems to find the best kernels overall.

In order to more quantitatively evaluate the performance of the CPEs trained the prediction error is used. A simple baseline to compare against is using the last shown image as the predicted output which gives a mean prediction error of 33.79 with a standard deviation of 30.85. Table 2.1 shows the mean and standard deviation of the prediction error on the last 1000 parameter updates for CPEs of varying max-pooling sizes.

max-pool size	1x1 (s)	2x2 (s)	3x3 (s)	4x4 (s)
1 (t)	19.06 (13.01)	20.43 (13.62)	20.28 (12.79)	21.53 (13.11)
2 (t)		24.93 (14.54)	26.01 (14.25)	26.70 (14.27)

Table 2.1: Mean prediction error (standard deviation), of CPE for varying sizes of spatial (s) and temporal (t) max-pooling.

All CPEs give better predictions than the baseline of just using the last input image as the prediction, which had a mean prediction error of 33.79. The standard deviation is relatively high on all CPEs, reflecting the variety of scenes in the dataset. It is evident that the temporal max-pooling severely limits the CPE ability to make accurate predictions. This could be simply due to the fact that it has less expressive power, but the CPE with 2x2x2 max pooling has more power than the CPE with 1x4x4 max pooling, and has worse prediction error. A single predicted frame from each model along with the frame to be predicted are shown below.

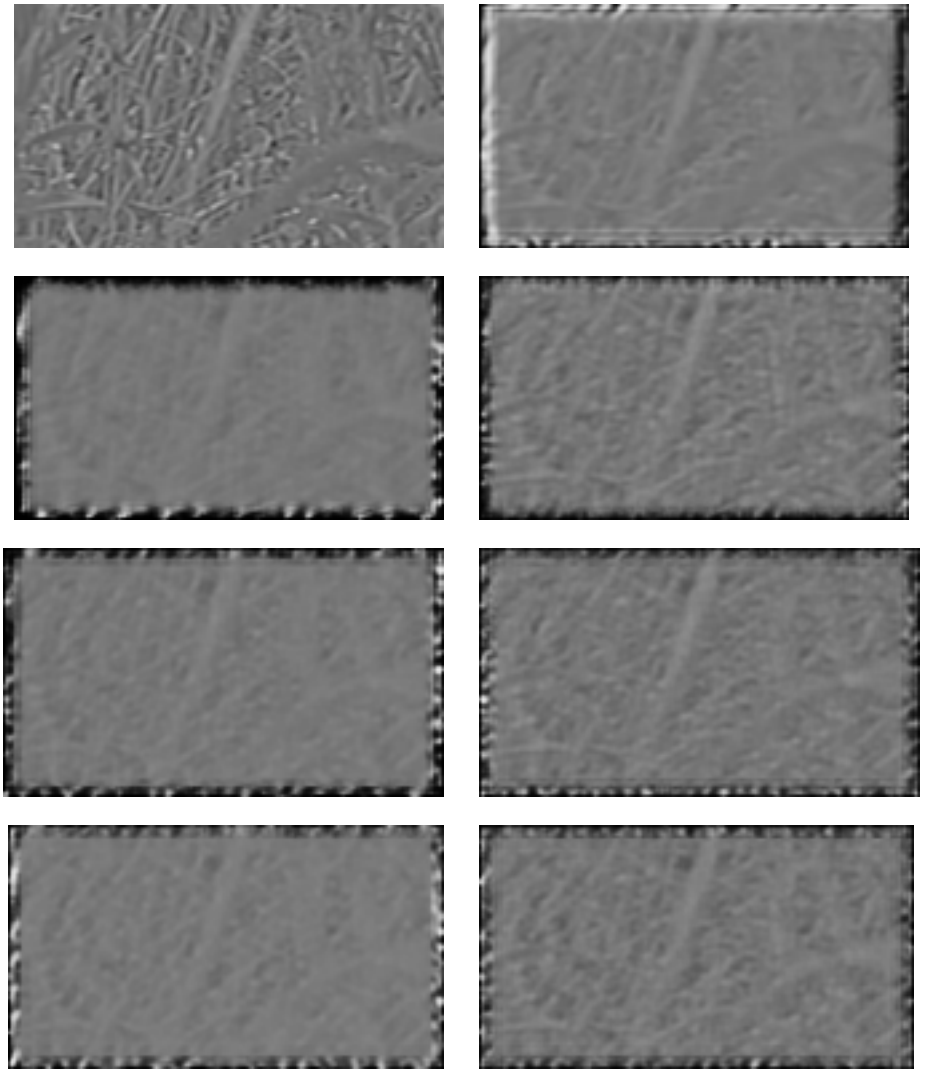

Figure 2.28: CPE predicted frames. Top left: frame to be predicted. From top-left to bottom right max pooling sizes: 1x1x1, 2x2x2, 1x2x2, 2x3x3, 1x3x3, 2x4x4, 1x4x4.

The CPE predictions looks like a blurred version of the real image. If one flickers rapidly between the target frame and the predicted frame, one will see that the predicted frame is actually translated a couple of pixels to the right. These frames are from a video that is slowly translating left, so what the CPEs have

done are to reconstruct a blurred version of the last input image they saw. In terms of prediction, a blurred version of the last input is not a bad strategy, but it is far from optimal. The blurred reconstruction somewhat contradicts the highly structured kernels found.

In summary the CPE was shown to learn some structured filters, resembling gabor filters and some noisy filters. The predictions made by the best CPEs seemed to be blurred reconstructions of the last input images they were shown. The discussion will evaluate and attempt to explain some of these findings.

Discussion

Promising findings

The prediction learning framework leads to structured filters, some resembling gabor filters

The predictive encoder and the convolutional predictive encoder trained on natural video have both been shown to lead to hidden representations with receptive fields responding to structured filters, some resembling oriented gabor filters. This is the strongest finding of this thesis, as it directly supports the hypothesis that a simple learning rule based on minimizing prediction error works to the degree that it can replicate receptive fields as those seen in visual area V1. Few learning paradigms exists that have found these filters, most notable are sparse coding [OF97] including all the models using sparsity constraints, e.g. [LGRN09, LZYN11], Independent Component Analysis [vHR98] and slow feature analysis [BW05]. This is a new paradigm, that while not perfect, has several attractive properties, is biologically plausible and has a solid theoretical foundation.

Minimizing prediction error is a general, unregularized, and simple learning rule.

The proposed architectures and learning algorithms have several attractive attributes. First of all the learning rule is general. It can be applied to any sensory data, including vision, sound, touch, taste which all have a temporal component and any imaginable man made temporal data such as weather readings (pressure, temperature, wind speeed, etc.), traffic flow, website usage, stock market trends, etc.

Another very attractive property is that there was no need to regularize the algorithm, e.g. using a sparsity constraint or limiting the hidden layer size. Instead of learning an easy task while severely limiting the model the prediction learning framework learns a hard task without imposing any constraints on the model. This reduces the amount of hyper-parameters and lends strength to the core hypothesis, that prediction alone can serve as a learning signal. As mentioned several other proposed models does not find good representations without being sparsity constrained. While their results are usually impressive, the need for a sparsity constraint shows that their core learning algorithm is not sufficient to find good representations. While the CPE did find better filters when max-pooled, the PE found good filters without any regularization with an overcomplete hidden representation. It is hypothesized that enough training of a CPE without max-pooling would lead to similar results.

In line with not requiring any regularization the learning algorithm and architectures are relatively simple. The learning algorithm is simply stochastic gradient descent on the prediction error, implemented computationally using the well known back-propagation algorithm. The proposed architectures are also relatively simple, basically being extensions into the temporal domain of an autoencoder and a convolutional autoencoder respectively. For the CPE the more advanced stochastic diagonal levenberg-marquardt learning algorithm was introduced to speed up learning, but this is not a required step, it merely reduces learning time.

The top-down predictions are useful and fit into the theoretical framework of biased attention.

The top down connections in the prediction learning framework are useful in two ways. First, being able to predict ones immediate sensory input is very useful for ones survival, such that one can avoid pain. If stacked up in a hierarchical manner, being able to predict high-level representations of the word might enable

one to predict longer-term effects of actions. From an evolutionary standpoint it is highly likely that genes enabling this kind of prediction would proliferate. It is important to note that while one may predict the long and short term effects of ones actions, the proposed framework does nothing to explain actually deciding which action to take, which implies understanding some notion of 'good' or 'bad'. But having a notion of 'good' or 'bad' without being able to predict the effects of ones actions would be equally useless. Secondly it is impossible to infer the correct configuration of the scene giving rise to the sensory inputs out of the infinite possible configurations giving rise to the same scene without having top-down connections [Fri03]. In this light the top-down connections disambiguate highly ambiguous sensory input, based on memory, higher order beliefs and previous sensory input. A further strength of the proposed framework is that the top-down predictions fits directly in to the larger theoretical frameworks of biased attention and predictive coding.

The prediction learning framework has been shown to scale to full size images

Scaling to realistic image sizes, i.e. 200x200 pixels remains a challenging task [LGRN09], and has been identified as a key challenge for the deep learning community by Andrew Ng [Ng09]. Scaling video to realistic sizes are even more challenging due to the extra dimension of data. The CPE was shown to find some good representations when trained on video of size 160x90 pixels. While this is not yet realistic size video it is fairly close and provides enough information to convey complex scenes. The use of a convolutional model was required to scale to this level.

The proposed models can be trained layerwise and stacked into deep architectures

The PE and CPE were specifically designed such that they could easily be stacked into deep architectures as this have been shown to be a promising way to learn deep hierarchical feature representations. In the case of the PE, the data would simply be transformed into the hidden representations and the second layer would learn with these as their visible states. The same holds true for the CPE, where the second layer CPE would learn on the activities of the feature cubes of the first layer CPE. So far experiments have been focused on learning a single layer of these models, in order to validate their effectiveness as a learning module. It remains to be shown whether they work well when stacked up.

Challenges

Several of the filters found are not gabor filters

The main problem is that far from all of the filters found are gabor filters. Most other models find one set of filters, i.e. blob detectors or oriented gratings or gabor filters. The PE finds a mix of all of these, which shows that the model is not restricted to one type of filters or the other, but instead can find a combination of filters most suitable to the task at hand. The CPE found several noisy filters so it might be prudent to look at the differences of the two models. The key difference is in the training. The CPE was trained for much fewer parameter updates, 20.000 compared to 10 million, using a much smaller learning rate 0.001 compared to 0.1, though due to the second order method and momentum, the maximum possible learning rate for the CPE was 0.004. Also every gradient step of the CPE the kernel weights were fitted to minimize prediction error over all image positions. This is comparable to training the PE with a mini batch size of approximately $(160 - 10 + 1)(90 - 10 + 1)(11 - 2 + 1) = 122.310$ which is quite unlike the mini batch size of 1 that was used. In effect the CPE could maximally change its weights $\frac{0.004\,\text{gradient}}{\text{step}} 20.000\,\text{step} = 80\,\text{gradient}$, on a "global" loss function while the PE could change its weights $\frac{0.1\,\text{gradient}}{\text{step}} 10 \times 10^6\,\text{step} = 1 \times 10^6\,\text{gradient}$ on a local loss function. Further the global loss function is hypothesized to be much flatter due to the averaging out of nearly all gradients, while the local loss function for a single batch will have steep parameter gradients minimizing the loss on just that patch. This might be one explanation as to why the PE found much better filters than the CPE. It is hypothesized that with more learning the CPE would find better kernels.

Sensitivity to max-pooling hyper-parameter

Sparsity constraints have been shown on numerous occasions to lead to gabor like filters in models trained on natural images. As such, using a sparsity constraint while investigating a new models ability to extract gabor like filters from natural data is not optimal as any positive results could be attributed to the sparsity constraint. For this reason the models proposed have been free of sparsity constraints, except for the max-pooling in the CPE. The max-pooling in the CPE was used as max-pooling was shown to lead to much better filters in the static case when training a CAE on MNIST. Unfortunately it also proved to lead to better filters in the CPE, thus making it hard to distinguish whether the good filters found were due to max pooling on a noisy reconstruction task, which prediction can be approximated as, or whether it was due to minimizing prediction error. It should be noted, that while the filters are not as good, the

CPE without max-pooling did find gabor like filters, especially on the input kernels. It is hypothesized that with enough training max-pooling becomes less needed.

Long training time

A major problem with the CPE was the prohibitively long training time. The final models, which were trained for just 20.000 parameter updates took approximately 70 hours to train. This made debugging, trying out hyper-parameters and testing hypothesises very time consuming and a lot less flexible and easy than desired. The training time for the PE was approximately 10 times less, reflecting that the CPE processed what corresponds to $20.000 \times 122.310 = 2.44 \times 10^9$ image patches, whereas the PE only processed 10×10^6 image patches. In this respect the CPE trained relatively faster, but the huge amount of training data slowed it down.

Future research

Use Spatio-Temporally whitened data

The single most interesting observation is that the data used was not temporally whitened. Spatial whitening is a sound pre-processing step which removes pixel-to-pixel correlations, usually found between neighbouring pixels to force the model to find more interesting features than blob detectors. But if the data is not temporally whitened the model can simply find temporal blob detectors and use those to good effect for reconstruction. It is hypothesized that the lack of temporal whitening has made the prediction task much easier, allowing the models to find less spatio-temporally interesting filters and that spatio-temporally whitening the data would lead to much better filters. That spatio-temporal whitening is a sound step is supported by the finding that the human visual system seems to perform this exact processing [CASB09]. Running the same experiments on spatio-temporally whitened data should be a top priority for future research.

Find a way to train the CPE more efficiently

As have been pointed out the training time for the CPE was excessive and the number of training iterations and the learning from each iteration too small.

Research should be directed at finding a training regime that uses less patches per training iteration, which would lower the training time per iteration allowing more iterations. Simple ideas include smaller images patches, i.e. 80x45 or 50x50 and smaller temporal batches, e.g. 3 images per input cube instead of 11. To scale up the model again the filters found on the smaller input cubes could simply be used on larger input cubes, which should work just as well as filters learned on bigger input cubes.

Use parallel computing

To further reduce the training time the computations should be done in parallel, ideally on a GPU or GPU cluster. Currently the computations were all run sequentially resulting in the long training times observed. The CPE is ideally suited for GPU computing as 99.99% of the computational time was spent doing convolutions, a task which can be done efficiently in parallel on a GPU, although many small convolutions are less efficiently calculated on a GPU than fewer larger ones [AUAS+12]. A simple idea to avoid several small convolutions is to pool them into one large convolution, and then separate out the results afterwards.

Evaluate the models quantitatively

If the models, after the proposed changes, consistently finds good representations, experiments should be carried out that can evaluate the performance of the models quantitatively. A natural first choice of measure would be the invariance measure proposed by Goodfellow [GLS+09], which measures a feature detectors selectivity and invariance to temporal changes. Using the models for classification should be the next step. Datasets like the KTH human actions dataset [SLC04] and Hollywood and Hollywood2 [MLS09] are both spatio-temporal and should serve as good benchmarks.

Investigate similarities with the denoising autoencoder

As previously mentioned the PE is comparable to a DAE, in which the added noise is the differences between successive frames. It would be interesting to thoroughly investigate these frame-to-frame statistics to determine if they could be approximated with some distribution, and if using this distribution for the corruption process in a DAE would lead to results similar to those found by the PE. This would be a good first start to answer whether the filter found by the

PE is a result of learning to de-noise the frame-to-frame difference distribution, or whether it is predicting the highly structured transformations taking place that leads to good filters.

Investigate similar models

Several models close to the PE and CPE exists that would be interesting to investigate. A first natural choice based on the conditional RBM is a PE or CPE with visible to visible linear connections. The visible to visible connections could model simple correlations which would free up the top-down connections to model higher order dependencies. In a similar vein, and inspired by predictive coding, the top-to-bottom connections could carry only the error signal, i.e. how wrong were the predictions made. The layer above would learn to model the errors made by the lower layer, effectively carrying exactly the information needed to make perfect predictions at lower levels in the top-down connections.

Appendix: One-Dimensional convolutional back-propagation

The core equations for the convolutional neural net

$$
\begin{aligned}
a(x) &= \operatorname{sigm}\left(b + \left(\sum_{n=1}^{N} i(x-n)\ \operatorname{ik}(n)\right)\right) \\
o(x) &= \operatorname{sigm}\left(c + \left(\sum_{m=1}^{M} a(x-m)\ \operatorname{ok}(m)\right)\right) \\
e(x) &= o(x) - y(x) \\
L &= \frac{\sum_{x=1}^{X} e(x)^2}{2}
\end{aligned}
$$

The partial derivative for c, the output bias.

$$\frac{\partial}{\partial c} L = \sum_{x=1}^{X} e(x) \ \frac{\partial}{\partial c} e(x)$$

$$\frac{\partial}{\partial c} L = \sum_{x=1}^{X} e(x) \ \frac{\partial}{\partial c} \left(o(x) - y(x) \right)$$

$$\frac{\partial}{\partial c} L = \sum_{x=1}^{X} e(x) \ o(x) \ \left(1 - o(x) \right) \ \frac{\partial}{\partial c} \left(c + \left(\sum_{m=1}^{M} a(x-m) \ \mathrm{ok}(m) \right) \right)$$

$$\frac{\partial}{\partial c} L = - \sum_{x=1}^{X} e(x) \ o(x) \ \left(o(x) - 1 \right)$$

$$\frac{\partial}{\partial c} L = \sum_{x=1}^{X} \mathrm{od}(x)$$

The partial derivative for ok, the output kernels. Notice that a' equals a flipped in all dimensions, i.e. reverse ordered.

$$\frac{\partial}{\partial \mathrm{ok}(m)} L = \sum_{x=1}^{X} \mathrm{od}(x) \ \frac{\partial}{\partial \mathrm{ok}(m)} \left(c + \left(\sum_{m=1}^{M} a(x-m) \ \mathrm{ok}(m) \right) \right)$$

$$\frac{\partial}{\partial \mathrm{ok}(m)} L = \sum_{x=1}^{X} a(x-m) \ \mathrm{od}(x)$$

$$\frac{\partial}{\partial \mathrm{ok}(m)} L = \sum_{x=1}^{X} a'(m-x) \ \mathrm{od}(x)$$

The partial derivative for b, the feature biases.

$$\frac{\partial}{\partial b} L = \sum_{x=1}^{X} \mathrm{od}(x) \; \frac{\partial}{\partial b} \left(c + \left(\sum_{m=1}^{M} a(x-m) \; \mathrm{ok}(m) \right) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x=1}^{X} \mathrm{od}(x) \; \left(\sum_{m=1}^{M} \frac{\partial}{\partial b} \left(a(x-m) \; \mathrm{ok}(m) \right) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x=1}^{X} \mathrm{od}(x) \; \left(\sum_{m=1}^{M} \frac{\partial}{\partial b} \mathrm{sigm} \left(b + \left(\sum_{n=1}^{N} i(x-n-m) \; \mathrm{ik}(n) \right) \right) \; \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x=1}^{X} \mathrm{od}(x) \; \left(\sum_{m=1}^{M} a(x-m) \; (1 - a(x-m)) \; \frac{\partial}{\partial b} \left(b + \left(\sum_{n=1}^{N} i(x-n-m) \; \mathrm{ik}(n) \right) \right) \; \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x=1}^{X} \left(\sum_{m=1}^{M} \mathrm{od}(x) \; a(x-m) \; (1 - a(x-m)) \; \mathrm{ok}(m) \right)$$

$$x' = x - m$$

$$\frac{\partial}{\partial b} L = \sum_{x'=1-m}^{X-m} \left(\sum_{m=1}^{M} \mathrm{od}(x'+m) \; a(x') \; (1 - a(x')) \; \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x'=1-m}^{X-m} \left(\sum_{m=1}^{M} \mathrm{od}(x'+m) \; a(x') \; (1 - a(x')) \; \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x'=1-m}^{X-m} a(x') \; (1 - a(x')) \; \left(\sum_{m=1}^{M} \mathrm{od}(x'+m) \; \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x'=1-m}^{X-m} a(x') \; (1 - a(x')) \; \left(\sum_{m=1}^{M} \mathrm{od}(x'-m) \; \mathrm{ok}'(m) \right)$$

$$\frac{\partial}{\partial b} L = \sum_{x'=1-m}^{X-m} \mathrm{ad}(x')$$

The partial derivative for ik, the input kernels.

$$\frac{\partial}{\partial \mathrm{ik}(n)} L = \sum_{x'=1-m}^{X-m} \left(\sum_{m=1}^{M} \mathrm{od}(x'+m)\ a(x')\ (1-a(x'))\ \frac{\partial}{\partial \mathrm{ik}(n)} \left(b + \left(\sum_{n=1}^{N} i(x'-n)\ \mathrm{ik}(n) \right) \right) \right) \mathrm{ok}(m)$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} L = \sum_{x'=1-m}^{X-m} \left(\sum_{m=1}^{M} \mathrm{od}(x'+m)\ a(x')\ (1-a(x'))\ i(x'-n)\ \mathrm{ok}(m) \right)$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} L = \sum_{x'=1-m}^{X-m} i(x'-n)\ \mathrm{ad}(x')$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} L = \sum_{x'=1-m}^{X-m} i'(n-x')\ \mathrm{ad}(x')$$

Appendix: One-Dimensional second order convolutional back-propagation

---

The core equations for the convolutional neural net

$$
\begin{aligned}
\mathrm{za}(x) &= b + \left(\sum_{n=1}^{N} i(x-n)\ \mathrm{ik}(n) \right) \\
a(x) &= \mathrm{sigm}(\mathrm{za}(x)) \\
\mathrm{zo}(x) &= c + \left(\sum_{m=1}^{M} a(x-m)\ \mathrm{ok}(m) \right) \\
o(x) &= \mathrm{sigm}(\mathrm{zo}(x)) \\
e(x) &= o(x) - y(x) \\
L &= \frac{\sum_{x=1}^{X} e(x)^2}{2}
\end{aligned}
$$

The partial derivative for any parameter h.

$$\frac{\partial}{\partial h} L = \sum_{x=1}^{X} e(x)\ \frac{\partial}{\partial h} o(x)$$
$$\frac{\partial^2}{\partial h^2} L = \sum_{x=1}^{X} \left(\frac{\partial}{\partial h} e(x)\ \frac{\partial}{\partial h} e(x) + e(x)\ \frac{\partial^2}{\partial h^2} e(x) \right)$$

Dropping the second order terms.

$$\frac{\partial^2}{\partial h^2} L = \sum_{x=1}^{X} \frac{\partial}{\partial h} o(x)^2$$
$$\frac{\partial^2}{\partial h^2} L = \sum_{x=1}^{X} \left(o(x)\ (1 - o(x))\ \frac{\partial}{\partial h} \mathrm{zo}(x) \right)^2$$

The partial derivative for c, the output bias.

$$\frac{\partial}{\partial c} \mathrm{zo}(x) = 1$$
$$\frac{\partial^2}{\partial c^2} L = \sum_{x=1}^{X} \left(o(x)\ (1 - o(x)) \right)^2$$
$$\frac{\partial^2}{\partial c^2} L = \sum_{x=1}^{X} \mathrm{od2}(x)$$

The partial derivative for ok, the output kernels.

$$\frac{\partial}{\partial \mathrm{ok}(m)} \mathrm{zo}(x) = a(x - m)$$
$$\frac{\partial}{\partial \mathrm{ok}(m)} \left(\frac{\partial}{\partial \mathrm{ok}(m)} L \right) = \sum_{x=1}^{X} \left(o(x)\ (1 - o(x))\ a(x - m) \right)^2$$
$$\frac{\partial}{\partial \mathrm{ok}(m)} \left(\frac{\partial}{\partial \mathrm{ok}(m)} L \right) = \sum_{x=1}^{X} \mathrm{od2}(x)\ a'(m - x)^2$$

The partial derivative for b, the feature bias.

$$\frac{\partial}{\partial b}\mathrm{zo}(x) = \frac{\partial}{\partial b}\left(c + \left(\sum_{m=1}^{M} a(x-m)\ \mathrm{ok}(m)\right)\right)$$

$$\frac{\partial}{\partial b}a(x-m) = a(x-m)\ (1-a(x-m))\ \frac{\partial}{\partial b}\mathrm{za}(x-m)$$

$$\frac{\partial}{\partial b}a(x-m) = a(x-m)\ (1-a(x-m))$$

$$\frac{\partial}{\partial b}\mathrm{zo}(x) = \sum_{m=1}^{M} a(x-m)\ (1-a(x-m))\ \mathrm{ok}(m)$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x=1}^{X}\left(o(x)\ (1-o(x))\ \left(\sum_{m=1}^{M} a(x-m)\ (1-a(x-m))\ \mathrm{ok}(m)\right)\right)^2$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x=1}^{X}\left(\sum_{m=1}^{M} o(x)\ (1-o(x))\ a(x-m)\ (1-a(x-m))\ \mathrm{ok}(m)\right)^2$$

$$x' = x - m$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x'=1-m}^{X-m}\left(\sum_{m=1}^{M} o(x'+m)\ (1-o(x'+m))\ a(x')\ (1-a(x'))\ \mathrm{ok}(m)\right)^2$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x'=1-m}^{X-m}\left(a(x')\ (1-a(x'))\ \left(\sum_{m=1}^{M} o(x'+m)\ (1-o(x'+m))\ \mathrm{ok}(m)\right)\right)^2$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x'=1-m}^{X-m}\left(a(x')\ (1-a(x'))\right)^2\ \left(\sum_{m=1}^{M}\mathrm{od2}(x'+m)\ \mathrm{ok}(m)^2\right)$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x'=1-m}^{X-m}\left(a(x')\ (1-a(x'))\right)^2\ \left(\sum_{m=1}^{M}\mathrm{od2}(x'-m)\ \mathrm{ok}'(m)^2\right)$$

$$\frac{\partial^2}{\partial b^2}L = \sum_{x'=1-m}^{X-m}\mathrm{ad2}(x')$$

The partial derivative for ik, the input kernels.

$$\frac{\partial}{\partial \mathrm{ik}(n)} \mathrm{zo}(x) = \frac{\partial}{\partial \mathrm{ik}(n)} \left(c + \left(\sum_{m=1}^{M} a(x-m)\ \mathrm{ok}(m) \right) \right)$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} \mathrm{zo}(x) = \sum_{m=1}^{M} a(x-m)\ (1 - a(x-m))\ \frac{\partial}{\partial \mathrm{ik}(n)} \mathrm{za}(x-m)$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} \mathrm{zo}(x) = \sum_{m=1}^{M} a(x-m)\ (1 - a(x-m))\ i(x-n-m)$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} \left(\frac{\partial}{\partial \mathrm{ik}(n)} L \right) = \sum_{x=1}^{X} \left(o(x)\ (1 - o(x))\ \left(\sum_{m=1}^{M} a(x-m)\ (1 - a(x-m))\ i(x-n-m) \right) \right)^2$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} \left(\frac{\partial}{\partial \mathrm{ik}(n)} L \right) = \sum_{x'=1-m}^{X-m} \mathrm{ad2}(x)\ i(x'-n)^2$$

$$\frac{\partial}{\partial \mathrm{ik}(n)} \left(\frac{\partial}{\partial \mathrm{ik}(n)} L \right) = \sum_{x'=1-m}^{X-m} \mathrm{ad2}(x)\ i'(n-x')^2$$

Bibliography

[AB03] Alessandra Angelucci and Jean Bullier. Reaching beyond the classical receptive field of V1 neurons: horizontal or feedback axons? *Journal of Physiology-Paris*, 97(2-3):141–154, March 2003.

[AUAS+12] Shams A. H. Al Umairy, Alexander S. Amesfoort, Irwan D. Setija, Martijn C. Beurden, and Henk J. Sips. On the Use of Small 2D Convolutions on GPUs Computer Architecture. volume 6161 of *Lecture Notes in Computer Science*, chapter 6, pages 52–64. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2012.

[Ben09] Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[BLP+07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Université De Montréal, and Montréal Québec. Greedy layer-wise training of deep networks. In *In NIPS*, 2007.

[BM98] Dean V. Buonomano and Michael M. Merzenich. CORTICAL PLASTICITY: From Synapses to Maps. *Annual Review of Neuroscience*, 21(1):149–186, 1998.

[Bou06] Jake Bouvrie. Notes on convolutional neural networks. Technical report, 2006.

[BTS+01] M. Bar, R. B. Tootell, D. L. Schacter, D. N. Greve, B. Fischl, J. D. Mendola, B. R. Rosen, and A. M. Dale. Cortical mechanisms specific to explicit visual object recognition. *Neuron*, 29(2):529–535, February 2001.

[Bun90] C. Bundesen. A theory of visual attention. *Psychological review*, 97(4):523–547, October 1990.

[BW05] Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6):579–602, July 2005.

[CASB09] John Cass, David Alais, Branka Spehar, and Peter J. Bex. Temporal whitening: transient noise perceptually equalizes the 1/f temporal amplitude spectrum. *Journal of vision*, 9(10), 2009.

[CBB11] Aarron Courville, James Bergstra, and Yoshua Bengio. Unsupervised models of images by spike-and-slab rbms. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1145–1152, New York, NY, USA, June 2011. ACM.

[Che10] B. Chen. Deep learning of invariant spatio-temporal features from video. 2010.

[CMS12] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. February 2012.

[Cre77] Otto D. Creutzfeldt. Generality of the functional structure of the neocortex. *Naturwissenschaften*, 64(10):507–517, October 1977.

[Dau85] John G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Opt. Soc. Am. A*, 2(7):1160–1169, July 1985.

[DD95] Robert Desimone and John Duncan. Neural Mechanisms of Selective Visual Attention. *Annual Review of Neuroscience*, 18(1):193–222, 1995.

[dVB93] J. de Villiers and E. Barnard. Backpropagation neural nets with one and two hidden layers. *Neural Networks, IEEE Transactions on*, 4(1):136–141, January 1993.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 2010.

[EGHP98] E. R. Ergenzinger, M. M. Glasier, J. O. Hahm, and T. P. Pons. Cortically induced thalamic plasticity in the primate somatosensory system. *Nature neuroscience*, 1(3):226–229, July 1998.

[FÖ91] Peter Földiák. Learning invariance from transformation sequences. *Neural Comput.*, 3(2):194–200, June 1991.

[Fri03] Karl Friston. Learning and inference in the brain. *Neural Networks*, 16(9):1325–1352, November 2003.

[FVE91] Daniel J. Felleman and David C. Van Essen. Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, 1(1):1–47, January 1991.

[GCR+96] J. L. Gallant, C. E. Connor, S. Rakshit, J. W. Lewis, and D. C. Van Essen. Neural responses to polar, hyperbolic, and Cartesian gratings in area V4 of the macaque monkey. *Journal of neurophysiology*, 76(4):2718–2739, October 1996.

[GHB07] H. T. Ghashghaei, C. C. Hilgetag, and H. Barbas. Sequence of information processing for emotions based on the anatomic dialogue between prefrontal cortex and amygdala. *NeuroImage*, 34(3):905–923, February 2007.

[GLS+09] Ian J. Goodfellow, Quoc V. Le, Andrew M. Saxe, Honglak Lee, and Andrew Y. Ng. Measuring Invariances in Deep Networks. 2009.

[Has86] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 6–20, New York, NY, USA, 1986. ACM.

[HB04] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, October 2004.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[HW62] D. H. HUBEL and T. N. WIESEL. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160:106–154, January 1962.

[HW68] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, March 1968.

[KGB07] Kestutis Kveraga, Avniel S. Ghuman, and Moshe Bar. Top-down predictions in the cognitive brain. *Brain and Cognition*, 65(2):145–168, November 2007.

[KM08] A. Klaser and M. Marszalek. A spatio-temporal descriptor based on 3d-gradients. 2008.

[Kri10] Alex Krizhevsky. Convolutional deep belief networks on cifar-10. Technical report, 2010.

[KSJ00] Eric Kandel, James Schwartz, and Thomas Jessell. *Principles of Neural Science*. McGraw-Hill Medical, 4 edition, January 2000.

[Lap05] I. Laptev. On space-time interest points. *International Journal of Computer Vision*, 64(2):107–123, 2005.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[LBOM98] Y. Lecun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient BackProp. In G. Orr and K. Müller, editors, *Neural Networks—Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 5–50. Springer Verlag, 1998.

[LCBD+90] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten digit recognition with a back-propagation network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[LGRN09] 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*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[LZYN11] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3361–3368. IEEE, June 2011.

[MH07] R. Memisevic and G. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[MH10] Mohamed and G. Hinton. Phone recognition using Restricted Boltzmann Machines. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 4354–4357. IEEE, March 2010.

[MLS09] Marcin Marszałek, Ivan Laptev, and Cordelia Schmid. Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009.

[MMC+09] Hossein Mobahi, Hossein Mobahi, Ronan Collobert, Ronan Collobert, Jason Weston, and Jason Weston. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 737–744, New York, NY, USA, 2009. ACM.

[MMCS11] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th international conference on Artificial neural networks - Volume Part I*, ICANN'11, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag.

[Mou97] V. B. Mountcastle. The columnar organization of the neocortex. *Brain : a journal of neurology*, 120 (Pt 4):701–722, April 1997.

[Ng09] Andrew Ng. Unsupervised discovery of structure, succinct representations and sparsity. In *International Conference on Machine Learning*, 2009.

[OF97] Bruno A. Olshausen and David J. Fieldt. Sparse coding with an overcomplete basis set: a strategy employed by V1. In *Vision Research*, pages 3311–3325, 1997.

[OF04] B. Olshausen and D. Field. Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, 14(4):481–487, August 2004.

[PCL06] Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2006*, 2006.

[QvdH05] Fangtu T. Qiu and Rüdiger von der Heydt. Figure and Ground in the Visual Cortex: V2 Combines Stereoscopic Cues with Gestalt Rules. *Neuron*, 47(1):155–166, July 2005.

[RB99] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, January 1999.

[Rin02] Dario L. Ringach. Spatial Structure and Symmetry of Simple-Cell Receptive Fields in Macaque Primary Visual Cortex. *Journal of Neurophysiology*, 88(1):455–463, July 2002.

[RS01] Rajesh P. N. Rao and Terrence J. Sejnowski. Spike-Timing-Dependent Hebbian Plasticity as Temporal Difference Learning. *Neural Computation*, 13(10):2221–2237, October 2001.

[SB95] P. A. Salin and J. Bullier. Corticocortical connections in the visual system: structure and function. *Physiological reviews*, 75(1):107–154, January 1995.

[SKS+05] Amir Shmuel, Maria Korman, Anna Sterkin, Michal Harel, Shimon Ullman, Rafael Malach, and Amiram Grinvald. Retinotopic axis specificity and selective clustering of feedback projections from V2 to V1 in the owl monkey. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 25(8):2117–2131, February 2005.

[SLC04] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local SVM approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36 Vol.3. IEEE, August 2004.

[SLD82] M. V. Srinivasan, S. B. Laughlin, and A. Dubs. Predictive coding: a fresh view of inhibition in the retina. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 216(1205):427–459, November 1982.

[SMA00] Sen Song, Kenneth D. Miller, and L. F. Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, September 2000.

[Spr08] M. W. Spratling. Reconciling Predictive Coding and Biased Competition Models of Cortical Function. In *FRONTIERS IN COMPUTATIONAL NEUROSCIENCE*, pages 1–8, 2008.

[Spr12] M. W. Spratling. Unsupervised learning of generative and discriminative weights encoding elementary image components in a predictive coding model of cortical function. *Neural computation*, 24(1):60–103, January 2012.

[Sto96] James V. Stone. Learning Perceptually Salient Visual Parameters Using Spatiotemporal Smoothness Constraints. *Neural Computation*, 8(7):1463–1492, October 1996.

[SWB+07] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 411–426, 2007.

[TB10] Graham W. Taylor and Chris Bregler. Learning local spatio-temporal features for activity recognition. *Learning*, 4:0–1, 2010.

[THR11] Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Two Distributed-State Models For Generating High-Dimensional Time Series. *Journal of Machine Learning Research*, 2011.

[vHR98] J. H. van Hateren and D. L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings. Biological sciences / The Royal Society*, 265(1412):2315–2320, December 1998.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 2010.

[WS02] Laurenz Wiskott and Terrence J. Sejnowski. Slow Feature Analysis: Unsupervised Learning of Invariances. 2002.

[YCC93] X. H. Yu, G. A. Chen, and S. X. Cheng. Acceleration of backpropagation learning using optimised learning rate and momentum. *Electronics Letters*, 29(14):1288–1290, July 1993.