CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Group Invariant Convolutional Boltzmann Machines

Master's thesis in Mathematics

Maria Lindström

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Thesis for the degree of Master of Science

Group Invariant Convolutional Boltzmann Machines

MARIA LINDSTRÖM

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

MARIA LINDSTRÖM

Supervisor: Daniel Persson, Department of Mathematical Sciences
Examiner: Klas Modin, Department of Mathematical Sciences

Master's Thesis 2020
Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

We investigate group invariance in unsupervised learning in the context of certain generative networks based on Boltzmann machines. Specifically, we introduce a generalization of restricted Boltzmann machines which is adapted to input data that is acted upon by any compact group G. This is done by using certain G-equivariant convolutions between layers. We prove that the deep belief networks constructed from such Boltzmann machines define probability distributions that are invariant with respect to the action of G.

Acknowledgements

I would like to take the opportunity to thank all of the people that have helped me complete this thesis. First of all I would like to thank my supervisor Daniel Persson who has guided me through this work and pointed me in the right direction whenever I needed. I would also like to thank my family and friends who has supported me during the process of writing this thesis.

Contents

1 Introduction

1.1 Machine Learning

Computers are great at solving problems involving complicated calculations. Some tasks that would be impossible for human brains to solve, the computer can solve easily. However, some tasks that would be useful to automize are almost impossible to tell a computer how to solve. For example, when you see a picture of a cat, you can recognize this cat immediately. If we tried to tell a computer how to recognize a cat in a picture, this would get complicated. We would need to tell the computer which pixel combinations represents a cat and which does not. We would have to consider all of the different angles the picture could be taken from, all of the different positions the cat could be in and so on. With classical programming techniques this task becomes impossible to solve.

So why is it that the human brain can solve this task easily, while the computer can not? The answer to this question is that the brain is an expert on finding patterns and learning from previous experiences. We learn to recognize things by being exposed to examples. Machine learning is a collective name for different algorithms that mimics this learning procedure.

If we want the computer to learn to recognize cats in images, using a machine learning algorithm would look something like this:

1. Collect a set of pictures, some with a cat in them and some without. For every picture, we also provide the right label, that is we give the image the label cat if it is present and the label no cat if there is no cat in the picture.
2. Define a function that takes the images as input and outputs a binary value representing the label cat or no cat.
3. For each picture, calculate the output of the function and compare the results with the true labels.
4. Evaluate how good the function is at predicting the true labels, and adjust it so that it will perform better.
5. Repeat step 3 and 4 until you find the optimal function.
6. Evaluate the function on pictures that have not been used in the training process to see if it actually performs well on new data.

The steps above are just a rough sketch of how a machine learning algorithm could work. It can look different depending on which task we want to solve, and each step is implemented differently depending on which algorithm we use. What all algorithms have in common is that a lot of training data is needed for the learning process. In the above example, the training data consists of the pictures together with the labels.

The field of machine learning has become more useful the last decades, since the amount of available data has exploded. This has lead to an increasing number of algorithms and more effective ways to train the models. Since new algorithms has developed so quickly, the mathematical understanding of the field has fallen behind. Hence it is now important to look at these new algorithms from a mathematical perspective. If we are able to develop a mathematical theory for machine learning, it will be easier to analyze new methods.

In this thesis we will explore a small part of the machine learning field from a mathematical perspective. The focus will be on deep learning algorithms. Deep learning is a collective name for machine learning algorithms that uses deep structures built up of layers. Specifically we will look at convolutional neural networks and deep restricted Boltzmann machines.

1.2 Convolutional Neural Networks

Neural networks are a branch of machine learning that is slightly inspired by how neurons are connected in the brain. It is built up by layers of so called neurons. Each of the neurons in the first layer, which represents the input, is connected to neurons in the next layer and so on. When a neuron is activated, it sends a signal through its connections to activate new neurons. These neurons will in their turn send a new signal through all their connections, this process will continue until we reach the last layer, which represents the output. A subgroup of neural networks is *convolutional* neural networks. In these networks, the connections between one layer of neurons and the next follow a certain pattern. The restrictions on the connections in convolutional neural networks forces them to account for the symmetry in the input data. Whenever the network has learned to find a feature in the input, it will find it no matter where in the input it lies. Once again think of the example of deciding if there is a cat in a picture. If the network has learned how to find a cat, convolution assures that it will find it no matter where in the picture the cat is. This property is in mathematical terms called equivariance.

The input in classical neural networks is often real values arranged in a grid, such as images. In this case the property of convolutional neural networks is called *translation* equivariance. This means that when the picture is moved, that is translated, the output is translated in the same way. This type of neural networks have been commonly used in for example object detection in images. When we have more complicated inputs, such a spherical images, it is not as straigtht forward to implement convolution. A lot of articles addressing the problem of implementing convolution in spherical neural networks have been published, for example [Cohen et al., 2018] [Esteves et al., 2018] [Kondor et al., 2018] [Coors et al., 2018]. For spherical neural networks, the movement of the input is no longer translations, but rotations instead. Both rotations and translations are special cases of group actions. For this reason, there have been some recent attempts to generalize the concept of neural networks to allow input acted on by a general group G [Kondor and Trivedi, 2018] [Cohen and Welling, 2016]. Using convolutional connections in such networks guarantees that the network is equivariant to actions of the group G.

1.3 Supervised vs. Unsupervised Learning

The algorithms used in machine learning can roughly be divided into two groups: supervised and unsupervised learning. The above example of finding cats in images is a supervised problem. In supervised learning we are interested in finding a mapping from input to output. In this setting, we have a set of data $\{\mathbf{x}_i\}_{i=1}^N$ as input, and for each $\mathbf{x}_i$ we are given a label y_i. The goal is to find a function that models the relationship between $\mathbf{x}_i$ and y_i. Examples of problems we can solve using supervised learning algorithms are

- Spam filtering. The input is the received mail and the output is binary spam/no spam.
- Object detection in images. The input is an image and the output is yes/no depending on if a certain object is present in the image or not.
- Regression, for example the input is a list containing information of a house and the output is the price for the house.

To describe what unsupervised learning can mean, let's consider another task. Imagine that you want the computer to be able to generate a new picture of a cat. An algorithm for this task could look something like:

1. Collect a set of pictures, all representing a cat.
2. Define a probability distribution over the space of pictures.
3. Evaluate, given the probability distribution defined in step 2, the probability of drawing the set of pictures collected in step 1.
4. Adjust the probability distribution so that the probability in step 3 is maximized.
5. Use some sampling algorithm to draw new samples from the resulting probability distribution. If the learning algorithm performs well, the samples should look like pictures of cats.

As with supervised learning, these steps could look different depending on which algorithm we use, and different types of tasks need different steps. Unsupervised learning is often a less well-defined problem compared to supervised learning. There is no correct output for each input, instead we want the model to describe the input data in some way. For example you might want to find clusters in the data, or you want to know which distribution the samples were drawn from. Often, the goal is in some sense to find the probability distribution that generated the training examples [Goodfellow et al., 2016]. Examples of what we can use unsupervised learning for is

- Clustering: Given a set of input data, we want to cluster the data to find out what different groups there are in it.
- Generating images/ image inpainting. Here the goal is to find patterns in the input images to be able to construct new images of the same type. When inpainting, we are given a image where some values are missing, for example the image might be blurry, and we want to fill in the missing data.

It is often easier to evaluate the performance of supervised algorithms. Since we are given the right label for each example, the output from the algorithm can be compared to the right output, which makes it quite easy to formulate a goal for the algorithm. We simply want to fit the parameters describing our function so that we minimize the error of the output. We can also divide the data we have into what is called train and test set. The training set is used during training, and the learned function can then be evaluated on the test set to see how well it performs on data not used during training.

The disadvantage of supervised learning is that it requires a lot of labeled data. When collecting data, it can sometimes be time-consuming to provide the right label for each example. Unsupervised learning does not need these labels. Further there are some problems that cannot be formulated so that we can use a supervised algorithm, while all supervised problems could be formulated as unsupervised [Goodfellow et al., 2016].

1.4 Boltzmann Machines

Neural networks is a class of machine learning algorithms often used for supervised problems. They can however be a tool for unsupervised learning as well. One example of an unsupervised algorithm that uses neural networks is the restricted Boltzmann machine (RBM). It can solve tasks such as image generation by modeling the distribution from which the training data was generated from.

RBMs only uses two layers of neurons, but there are ways one can connect them to create deeper networks.

Since the Boltzmann machine uses neural networks, the equivariance property of convolutional neural networks will affect the properties of the Boltzmann machine. There have been a number of articles that implement convolution in the RBM and the deep networks that can be built by RBM's to [Lei et al., 2014] [Lee et al., 2009a] [Lee et al., 2009b] [Wang, 2018] [Norouzi et al., 2009]. Hence it is now interesting to see how the generalization of convolution and equivariance that was introduced for neural networks transfers to the RBM.

1.5 Main Result

The purpose of this thesis is to explore how generalized convolution can be used in restricted Boltzmann machines. The main result of this thesis is that we prove the following theorem:

Theorem 1. A G-convolutional restricted Boltzmann machine is invariant to actions of G.

Further we will prove that the same result holds for the deeper algorithms built on the RBM.

Theorem 2. Using G-convolution in all layers guarantees both the deep restricted Boltzmann machine and the deep belief network to be G-invariant.

1.6 Layout of Thesis

The thesis will be structured in the following way:

- We will begin with a basic introduction of classical neural networks and how to train them. Convolutional neural networks will be introduced.
- We present a generalization of neural networks and convolution that allows input data acted on by any group G. We also show that these G-convolutional neural network are guaranteed to be equivariant with respect to actions of G.
- The restricted Boltzmann machine is introduced, and we describe how the training for these machines is done. We also discuss how these machines can be connected to create the deep Boltzmann machine and the deep belief network.
- We generalize the restricted Boltzmann machine and prove that using convolution in such machine guarantees invariance. We do the same for both the deep Boltzmann machine and the deep belief network.

2 Basics of Neural Networks

In this chapter, we will introduce neural networks. To begin with, we will explain how a neural network is built, and what parameters are used in it. Then we will discuss the training procedure of these networks. An algorithm called backpropagation will also be described, this is often used when tuning the parameters of the network in the training process.

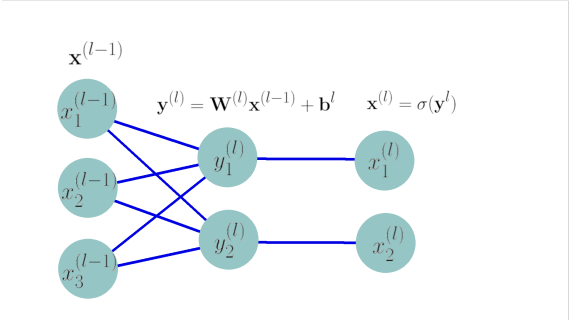

Figure 1: One layer with input $\mathbf{x}^{(l-1)} \in \mathbb{R}^3$ and output $\mathbf{x}^{(l)} \in \mathbb{R}^2$

2.1 Neural networks

Neural networks are a class of machine learning algorithm that models a function by composing many simple functions into a more complex one. Assume that our data set contains examples $\{\mathbf{x}_i, \mathbf{y}_i\}$ and we want to model the relationship between $\mathbf{x}_i$ and $\mathbf{y}_i$. A simple way to model the relationship between $\mathbf{x}$ and $\mathbf{y}$ is to use a linear model. This is however only useful when the true relationship is approximately linear. When we have a more complex relationship, we have to use more complex models, and a neural network is one way to do this. A neural network use compositions of linear functions, together with non-linearities between these function. The more such compositions we have in the network, the more complex function it can represent. This gives a chain of mappings $\mathbf{x}^{(0)} \mapsto \mathbf{x}^{(1)} \mapsto \ldots \mapsto \mathbf{x}^{(L)}$, and each of these $\mathbf{x}^{(l)}$'s is referred to as a *layer*.

Assume that the input $\mathbf{x}^{(l-1)}$ to the layer is a vector over some field, for simplicity we can take $\mathbf{x}^{(l-1)} \in \mathbb{R}^n$ for some n. A layer is a composition of two functions. The first function is linear, so the output is $\hat{\mathbf{x}}^{(l)} \in \mathbb{R}^m$ where $\hat{\mathbf{x}}^{(l)} = W\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$ for some matrix $W \in \mathbb{R}^{n\times m}$. The second function is nonlinear, often referred to as an activation function. Since the composition of two linear functions is still linear, the activation function is necessary to increase the complexity of the network for each layer. This activation function, lets call it σ^l, has to be differentiable and will be taken pointwise over the vector $\hat{\mathbf{x}}^{(l)}$. The resulting output is $\mathbf{x}^l$ where $x_j^l = \sigma(\hat{x}_j^l)$. We will use $\mathbf{x}^l = \sigma(\hat{\mathbf{x}}^l)$ for short. Summarizing this, we have that one layer is a function $\psi : \mathbb{R}^n \to \mathbb{R}^m$ defined as

$$\psi(\mathbf{x}^{l-1}) = \sigma^l(W^l\mathbf{x}^{l-1} + \mathbf{b}^l) \tag{1}$$

where $W^l \in \mathbb{R}^{n\times m}$ and $\mathbf{b}^l \in \mathbb{R}^m$. Figure 1 shows how a layer could look. What we have described here as a layer is the function that takes us from $\mathbf{x}^{(l-1)}$ to $\mathbf{x}^{(l)}$, sometimes we will instead refer to the vector $\mathbf{x}^{(l)}$ when we say layer l. Whether we mean the function or the vector when using the term layer should be clear from the context.

In a neural network, layers are connected to create a deep structure. Assume that the input to the network is $\mathbf{x}^{(0)} \in \mathbb{R}^{n_0}$, then we first map it to another vector $\mathbf{x}^{(1)} = \psi(\mathbf{x}^{(0)}) \in \mathbb{R}^{n_1}$, where ψ is defined as in equation (1). After this, $\mathbf{x}^{(1)}$ is mapped to another vector $\mathbf{x}^{(2)} \in \mathbb{R}^{n_2}$ in the same way and so on. If we use L to denote the number of layers we continue until we reach $\mathbf{x}^{(L)} \in \mathbb{R}^{n_L}$, which is the output of the network. The choice of dimension n_l for each layer is part of the design of the model, but n_0 and n_L must match the input and the output space respectively. This process defines a function $\mathbf{x}^{(0)} \mapsto \mathbf{x}^{(L)}$, and this is what we call an artificial neural network. Figure 2 shows how one can picture such a neural network.

The network described above is the simplest type of a neural network. It is called feed forward, meaning that each layer gets its input only from the previous layer, and we have restricted the input and output to be real valued vectors.

Definition 1 (Classical definition of feed forward neural network.)**.** A real feed forward neural

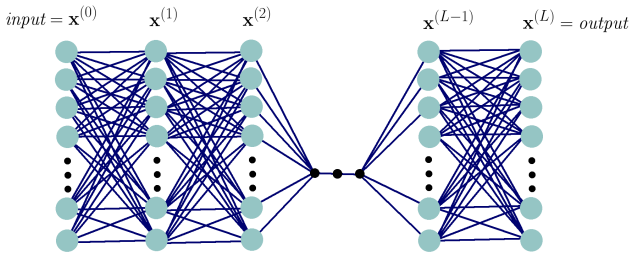

Figure 2: A neural network with L layers. The dots represents an arbitrary number of neurons and layers.

network is a function $\mathcal{N} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}, \mathcal{N}(\mathbf{x}^0) = \mathbf{x}^L$. The mapping from $\mathbf{x}^0 \mapsto \mathbf{x}^L$ is a composition of mappings $\mathbf{x}^{l-1} \mapsto \mathbf{x}^l$

$$\mathbf{x}^l = \psi(\mathbf{x}^{l-1}) = \sigma^l(W^l\mathbf{x}^{l-1} + \mathbf{b}^l)$$

where $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{b}^l \in \mathbb{R}^{n_l}$ and $\sigma^l : \mathbb{R} \to \mathbb{R}$ is a nonlinear differentiable function taken pointwise over its input.

A remark on the definition above: Neural networks are often not vector shaped. In many cases we have matrix or even tensors as the input and output of each layer. However, assume that the input to the layer is a matrix $X^{(l)} \in \mathbb{R}^{m_l \times n_l}$, then this matrix can be flattened out to a vector $\mathbf{x}^{(l)} \in \mathbb{R}^{m_l \cdot n_l}$. So the definition above still holds.

2.1.1 Parameters of a neural network

We now have defined a class of functions, called neural networks. To specify such function, all of its parameters have to be decided.

- L: We have to decide on the number of layers the network should have.
- n_l: For each $0 \leq l \leq L$ we have to decide the size of layer l.
- σ_l: For each $0 < l \leq L$ we have to decide which activation function to use.
- W^l and $\mathbf{b}^l$: For each layer $0 < l \leq L$ we have to decide the coefficients for the linear operator.

L determines the complexity of the functions the network will represent. We say that the network gets *deeper* when L gets larger, and the deeper the network is, the more complex will the function be. n_0 and n_L has to match the input and output space of the network, which should be clear from the problem you want to solve. For $0 < l < L$, n_l plays a part in deciding the complexity of the network, these parameters together with L will determine the whole size of the network. σ_l can be any differentiable nonlinear function, one common choice is the sigmoid function $\sigma = \frac{e^x}{e^x+1}$, but the choice of these functions is a part of designing the model. The activation function does not need to be the same in every layer.

The parameters we have described so far is set during the design of the model. When these parameters have been chosen, the only thing left to define the resulting model is W^l and $\mathbf{b}^l$. These parameters are found by training the model on a set of examples.

2.2 Training a neural network.

When training a neural network, we want to adjust the parameters W^l and $\mathbf{b}^l$ so that the function that the model represents is as close to the function that want to approximate, as possible. How this training should be done depends on what type of problem we want to solve, but in this section we will describe one type of training algorithm. Assume that the data set contains one feature that we want to be able to predict given the values of the other features, then we can pose the problem as: given $\mathbf{x}$, we want to be able to predict $\mathbf{y}$. That is, we want $\mathcal{N}(\mathbf{x}_i) = \mathbf{y}_i$. For example if the goal is to determine if there is a cat in a set of pictures or not, we have $\mathbf{x}$ is an image and $\mathbf{y} \in \{0, 1\}$, where 0 represents no cat and 1 represent cat. Then we want the network to output the true value for as many pictures as possible. Since $\mathcal{N}$ depends on the parameters in $\theta = \{W^l, ^l, l = 1, \ldots, L\}$, we can see θ as an argument to $\mathcal{N}$ as well as $\mathbf{x}$.

To be able to adjust the parameters $\theta = \{W^1, \mathbf{b}^1, \ldots, W^L, \mathbf{b}^L\}$ so that $\mathcal{N}(\mathbf{x}_i; \theta)$ is close to y_i for all i, we have to define what close means. We have to define a function $\mathcal{L}$ which takes the network and the training data as input and determines the quality of the predictions from the neural network.

The most straightforward way to define this function is to take the sum of squares of the distance between the output of the network and the values of y:

$$\mathcal{L}(\mathcal{N}, \theta, \{\mathbf{x}_i, \mathbf{y}_i\}) = \sum_{i=1}^{n} ||\mathcal{N}(\mathbf{x}_i, \theta) - \mathbf{y}_i||^2.$$

Since the function $\mathcal{N}$ depends on the parameters in θ, we can see them as an argument to $\mathcal{N}$. The values of $\mathbf{x}_i$ and y_i is fixed so the only thing varying is θ. Thus the problem is to minimize the function $\mathcal{L}$ with respect to θ.

θ will usually contain to many variables to solve this optimization problem analytically. Instead an algorithm based on gradient descent, called the *backpropagation algorithm*, will be used.

2.3 Backpropagation

For simplification, will assume that we train the model using only one example $\mathbf{x}$ during training. When adjusting the parameters θ to minimize the loss function $\mathcal{L}(\mathcal{N}(\mathbf{x}, \theta), \mathbf{y})$ stochastic gradient decent is used. This means that the parameters are updated stepwise. In each step, θ is updated with $\theta = \theta - \Delta_\theta$ where $\Delta_\theta = \lambda \nabla_\theta \mathcal{L}$ for some constant λ. This can be seen as $\theta_i = \theta_i - \lambda \frac{\partial \mathcal{L}}{\partial \theta_i}$. To efficiently calculate $\frac{\partial \mathcal{L}}{\partial \theta_i}$, we use an algorithm called backpropagation [Goodfellow et al., 2016]. This algorithm uses that the network is a composition of simple functions, together with the chain rule. The chain rule tells us that if $z = f(y)$ and $y = g(x)$, then derivative of z with respect to the variable x can be calculated as $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$. To simplify notation, let $\hat{y} = \mathcal{N}(\mathbf{x}, \theta)$, then we can write the loss function as $\mathcal{L}(\hat{y}, \mathbf{y})$. The derivative of the loss with respect to a variable θ_i can then be written as $\frac{\partial \mathcal{L}}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i}$. This gives the idea to go backwards in the graph, calculating the derivative of the current layer and then sending this function backwards.

Assume that we have a network $\mathcal{N}$ with L hidden layers, let $\sigma^{(l)}$ be the activation at layer l. θ then contains the parameters $\{W^{(l)}, b^{(l)} : l = 1, \ldots L\}$. Given a certain input $\mathbf{x}$, we get a sequence of layers $h^{(1)}, \ldots, h^{(L)}$. For each layer denote $a^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$, so that $h^{(l)} = \sigma^{(l)}(a^{(l)})$. At last, let $\mathcal{L}$ be the loss function we want to minimize. To update the parameters through stochastic gradient descent, we need to find $\nabla_{W^{(l)}} \mathcal{L}$ and $\nabla_{b^{(l)}} \mathcal{L}$ for all l. Then we update the parameters $W^{(l)}, b^{(l)}$ with $W^{(l)} = W^{(l)} - \lambda \nabla_{W^{(l)}} \mathcal{F}$ and $b^{(l)} = b^{(l)} - \lambda \nabla_{b^{(l)}} \mathcal{F}$. λ is a constant that decides how large steps will be taken in each iteration of the gradient descent.

The backpropagation starts from the loss function and works backward through the network. The

first step is to calculate the gradient of the loss function with respect to the last layer in the network: $\nabla_{h^{(L)}}\mathcal{F} = \nabla_{\hat{y}}\mathcal{F}$. Then for each $l = L, \dots, 1$ we can use the gradient $\nabla_{h^{(l)}}\mathcal{F}$ to calculate the gradients $\nabla_{W^{(l)}}\mathcal{F}, \nabla_{b^{(l)}}\mathcal{F}$ and $\nabla_{h^{(l-1)}}\mathcal{F}$. Then we can use $\nabla_{h^{(l-1)}}\mathcal{F}$ to calculate the gradients of the previous layer and can thus work our way back to the first layer. Next, we will explain how these gradients are calculated.

First, we will have to use that each layer is a composition of two functions: $h^{(l)} = \sigma^{(l)}(a^{(l)})$, where $a^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)}$. $a^{(l)}$ can thus be seen as a function of both $W^{(l)}, b^{(l)}$ and $h^{(l-1)}$. Using the chain rule for calculating any of the gradients we want, we will need the gradient $\nabla_{a^{(l)}}\mathcal{F}$. Since $\sigma^{(l)}$ is taken pointwise, we get that

$$\frac{\partial \mathcal{F}}{\partial a_i^{(l)}} = \sum_{j=1}^{n_l} \frac{\partial \mathcal{F}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial a_i^{(l)}} = \frac{\partial \mathcal{F}}{\partial h_i^{(l)}} \sigma^{(l)\prime}(a_i^{(l)}). \tag{2}$$

If we let $\sigma^{(l)\prime}(a^{(l)})$ be the vector containing all the derivatives $\sigma^{(l)\prime}(a_i^{(l)})$, we can rewrite equation 2 as

$$\nabla_{a^{(l)}}\mathcal{F} = \nabla_{h^{(l)}}\mathcal{F}\sigma^{(l)\prime}(a^{(l)}).$$

Now that we have $\nabla_{a^{(l)}}\mathcal{F}$, we can continue to calculate $\nabla_{W^{(l)}}\mathcal{F}$. Using the chain rule once again, and that $a_i^{(l)} = \sum_{j=1}^{n^{l-1}} W_{i,j}^{(l)} h_j^{(l-1)} + b_i^{(l)}$ gives

$$\frac{\partial \mathcal{F}}{\partial W_{i,j}^{(l)}} = \sum_{k=1}^{n^l} \frac{\partial \mathcal{F}}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial \mathcal{F}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial \mathcal{F}}{\partial a_i^{(l)}} h_j^{(l-1)}.$$

Hence the parameters $W^{(l)}$ can be updated with

$$\nabla_{W^{(l)}}\mathcal{F} = \nabla_{a^{(l)}}\mathcal{F}h^{(l-1)T}.$$

It is also easy to see that

$$\nabla_{b^{(l)}}\mathcal{F} = \nabla_{a^{(l)}}\mathcal{F}.$$

Now, to go back in the network and calculate the gradients of the previous layer, we have to be able to calculate $\nabla_{h^{(l-1)}}\mathcal{F}$. Since

$$\frac{\partial \mathcal{F}}{\partial h_j^{(l-1)}} = \sum_{i=1}^{n^{(l)}} \frac{\partial \mathcal{F}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial h_j^{(l-1)}} = \sum_{i=1}^{n^l} \frac{\partial \mathcal{F}}{\partial a_i^{(l)}} W_{i,j}^{(l)},$$

we have that

$$\nabla_{h^{(l-1)}}\mathcal{F} = \nabla_{a^{(l)}}\mathcal{F}W^{(l)}.$$

We can continue to calculate the gradients with respect to the parameters in the previous layer and so on until we reach layer 1. After all parameters have been updated, one step of the gradient descent is done. Then we use the updated parameters to once again go forward in the network to calculate the loss function, which in turn will be used to calculate the gradients of all parameters. This process continues until we reach some stopping criteria.

3 Convolutional Neural Networks

The neural networks that will be of interest in this thesis are *convolutional* neural networks. In this chapter we will first describe classical convolutional networks and translation equivariance. Then we will generalize convolution and neural networks to create networks that can be equivariant to actions of an arbitrary compact group.

3.1 Convolution and equivariance

A classical neural network is said to be convolutionan if it has at least one convolutional layer:

Definition 2 (Convolutional layer)**.** A convolutional layer is a layer in a neural network with the special form

$$x_j^{(l)} = \sigma(\sum_{i=1}^{k} x_{i+j-1}^{(l-1)} c_i^{(l)}) \text{for } j \in \{1, 2, \dots, n-k+1\} \tag{3}$$

for some constants $c_1^{(l)}, \dots, c_k^{(l)}$ and activation function σ.

This operator requires less parameters than the ordinary matrix multiplication used in a layer. The number of parameters in an ordinary layer is $(m+1) \cdot n$, where m is the size of the input vector and n the size of the output vector. In a convolutional layer, all elements in the output vector shares the same weights $\{c_i\}_{i=1}^k$, and they are not connected to every input element. Thus, there are only k parameters to optimize, which makes convolutional neural networks more efficient to train.

Except for the reduction of computation time, convolutional layers has another important property. This property is called *translation equivariance* and can be beneficial for many tasks. For example, the goal of the network could be to decide if a certain object is present in a picture. In this case it does not matter where in this picture the object might be, so you want the network to treat every part of the picture in the same way. If the network treats every part of the picture in the same way, then the network can detect the object no matter where it is positioned. It can be shown that, if each layer in a neural network is convolutional, then the whole network is translation equivariant. This is the main reason why convolutional neural networks have become so popular for solving certain problems.

Translations are an example of so called group actions. This leads us to the question if it is possible to build networks that are equivariant to other types of group actions. The networks described so far have been restricted to take input in $\mathbb{R}^n$, and we have only discussed translations acting on this space. For some applications, we want to draw conclutions from data formed in another way than a euclidean space, and in these case the actions can be of other forms. One example is when we have spherical input, for example spherical images. When talking about movement on the sphere, this is called rotations. Rotations are another kind of group actions, and recently there have been attempts of building networks that are equivariant to rotations of the input[Cohen et al., 2018]. We will give a summary of how they used convolutional neural networks on this type of inpu. First, we will give a more formal definition of group actions and equivariance.

3.2 Group Theory

Before explaining what group actions is, we have to define what is meant by a compact group.

Definition 3 (Group)**.** A group $(G, *)$ is a set G equipped with an operator $*$ that satisfies the following conditions:

- $a, b \in G \implies a * b \in G$.
- $(a * b) * c = a * (b * c)$.
- There exist $e \in G$ such that $e * a = a * e = a$ for all $a \in G$. e is called the identity element.

- For every $a \in G$ there exists $b \in G$ such that $a * b = e$, this is denoted as $a^{-1} = b$ and is called the inverse of a.

Definition 4 (Topological group)**.** Let $(G, *)$ be a group. If G is a topological space such that the group operator $*$ is continuous, we call G a *topological group.*

Definition 5 (Compact group)**.** A topological group $(G, *)$ is *compact* if the topological space G is compact.

It is often understood from the context what the operator $*$ is, and in this case we refer to the group $(G, *)$ only by G without mentioning which operator the set is equipped with. We also write ab instead of $a * b$ to simplify the notation.

Now if we look at the group G and pick some element $g \in G$, then the group action together with this g induces a function $T_g : G \to G$ by $T_g(h) = gh$. In this way we can identify the group G with the set of such functions $\{T_g : g \in G\}$, often we simplify the notation and just write $g(h)$ instead of $T_g(h)$. We say that the group G acts on the set G.

In the same way, we can have a group action of G on another set χ, such that for any $g \in G$, there is a function $T_g : \chi \to \chi$. This must satisfy the following two axioms

- $T_g \circ T_h = T_{gh}$.
- $T_e(x) = x$ for all $x \in \chi$, where e is the identity element in G.

If we have a group G, each element $g \in G$ defines a function $T_g : G \to G$, where $\mathbb{T}_g(x) = gx$. We say that the group G acts on the set G. We can in the same way let G act on other sets.

Definition 6. Let G be a group and χ a set. If, for all $g \in G$ there exists a function $T_g : \chi \to \chi$ that satisfies

- $T_g \circ T_h = T_{gh}$, and
- $T_e(x) = x$ for all $x \in \chi$, where e is the identity element in G,

we say that G acts on χ by $\{T_g\}_{g \in G}$.

Definition 7. Whenever a group G acts on a set $\mathcal{X}$, this induces an action of G on the space $\mathcal{L}_{\mathbb{V}}(X) = \{f : \chi \to \mathbb{V}\}$, where $\mathbb{V}$ is an arbitrary vector space. This induced action is defined as $\{\mathbb{T}_g\}_{g \in G}$ where $\mathbb{T}_g(f)(x) = f(T_g^{-1}(x))$.

One of the main benefits of using convolution in neural networks is that it guarantees that the network is equivariant to group actions. We are now ready to give a formal definition of what is meant by equivariance

Definition 8 (Equivariance)**.** Let G be a group that acts on the sets $\mathcal{X}_1$ and $\mathcal{X}_2$ with the actions $\{T_g^1\}, \{T_g^2\}$ respectively. We say that a function $f : \mathcal{X}_1 \to \mathcal{X}_2$ is equivariant to the action of G if

$$f(T_g^1(x)) = T_g^2(f(x)) \text{ for all } g \in G, x \in \mathcal{X}_1$$

We will need one last definition before introducing the new neural networks.

Definition 9. A group G is said to act transitively on the set X if, for any $x_0, x \in X$ there exists $g \in G$ such that $x = T_g x_0$. If G acts transitively on the space X we say that X is a homogeneous space of G.

Remark. If X is a homogeneous space of G, then $X \cong G/H$ for some subgroup H of G.

3.3 Generalizing Neural Networks

The rest of this chapter will be devoted to building up the theory for convolutional neural networks that are equivariant with respect to actions of *any* compact group. We will follow the reasoning of [Kondor and Trivedi, 2018].

To begin with, we will look at the neural networks we have already defined from a new perspective.

A vector $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, can be seen as a function $f : \mathbb{Z} \to \mathbb{R}$ where

$$f(i) = \begin{cases} x_i & \textit{if } i \in \{1, \dots, n\} \\ 0 & \textit{otherviwe.} \end{cases}$$

In the same way a matrix $W \in \mathbb{R}^{n \times m}$ can be viewed as a function $\Phi : \mathbb{Z}^2 \to \mathbb{R}$

$$\Phi(i, j) = \begin{cases} W_{i,j} & \textit{if } i \in \{1, \dots, n\}, j \in \{1, \dots, m\} \\ 0 & \textit{otherwise.} \end{cases}$$

Define the operator $\cdot$ between functions $f, g : \mathbb{Z} \to \mathbb{R}$ as

$$g \cdot f = \sum_{i \in \mathbb{Z}} g(i) f(i).$$

If g and f are functions representing vectors, the value of this operator will be the same as the value when taking the dot product between the two corresponding vectors.

We also want the analogue of matrix multiplication, thus define $\cdot$ between a function $\Phi : \mathbb{Z}^2 \to \mathbb{R}$ and f as above as $f \cdot \Phi : \mathbb{Z} \to R$

$$(f \cdot \Phi)(j) = \sum_{i \in \mathbb{Z}} f(i) \Phi(i, j). \tag{4}$$

If f represents a vector in $\mathbb{R}^m$ we have $(\Phi \cdot f)(i) = \sum_{j \in \mathbb{Z}} (\Phi(i, j) f(j)$. Let $L_V(X)$ denote the space of functions $\{f : X \to V\}$. Then the following definition is equivalent to definition 1:

Definition 10 (Feed Forward Neural Network.)**.** A real feed forward neural network is a function $\mathcal{N} : L_{\mathbb{R}}(\mathbb{Z}) \to L_{\mathbb{R}}(\mathbb{Z}), \mathcal{N}(f^0) = f^L$ where the mapping from $f^0 \mapsto f^L$ is a composite of mappings

$$f^0 \mapsto f^1 \mapsto \dots \mapsto f^L.$$

Each of the $f^{(l)}$'s lies in $L_{\mathbb{R}}(\mathbb{Z})$ and $f^{(l)}(i) = 0$ unless $i \in \{1, \dots, n_l\}$ for some finite n_l. Each of the mappings $f^{l-1} \mapsto f^l$ has the form:

$$f^l = \sigma^l(\Phi^l \cdot f^{l-1} + h^l)$$

where $\Phi^l \in L_{\mathbb{R}}(\mathbb{Z}^2)$, $h^l \in L_{\mathbb{R}}(\mathbb{Z})$ and $\sigma^l : \mathbb{R} \to \mathbb{R}$ is a nonlinear differentiable pointwise function.

So far, the only thing that has changed is notation. From this new point of view, we can see this definition as a special case of a more general definition. The input to our network is a function $f^{(0)} \in L_{\mathbb{V}}(X)$ for some space X, which will be referred to as an index set, and a vector space $\mathbb{V}$. This function is mapped to a function $f^{(1)}$ in the same space and so on, in the above definition $X = \mathbb{Z}$ and $\mathbb{V} = \mathbb{R}$, but in our general definition of a neural network they will be arbitrary. In this example all layers are functions over the same space $\mathbb{Z}$, but for some applications we will need to allow this space to vary between the layers. To further see how we should generalize our networks in a proper way, let us look at one example.

Example. In [Cohen et al., 2018] a convolutional network with spherical input was built. Here the input to the network is a function $f : S^2 \to \mathbb{R}$. The group that can act on the sphere is the rotation group $SO(3)$. There are two main issues when dealing with rotations on the sphere that differs from translations on the plane. First, even though the plane itself is a continuous group, we can easily approximate it with a discrete group by using a grid, keeping the symmetry of the translations. In this way, both the index set and the translations are isomorphic to $\mathbb{Z}^2$. When dealing with the sphere, there is no good way to approximate it in a discrete way that keeps the symmetry of rotations. The second issue is that in contrast to planar pictures, where the group actions and the index set is the same, the group of rotations on the sphere is not isomorphic to the sphere itself.

From this example we can see that we have to account for cases where the group acting on the index set is not isomorphic to the index set. However we will restrict the set X to be a homogeneous space of G. Whenever X is a homogeneous space of G, we can view X as a quotient space of G, so $X = G/H$ for some subgroup $H \in G$. For the space S^2 and $SO(3)$ we have the relation $S^2 = SO(3)/SO(2)$ where $SO(2)$ is the group of rotations in two dimensions.

Since the index set X will be a homogeneous space of G, we have that $X \cong G/H$ for some subgroup H of G. X can thus, for a fixed G, be defined by choosing the subgroup H instead. When we want to put the emphasis on the subgroup H, X will be replaced with G/H. Another change we will do in our new definition is that the mapping $f^{l-1} \mapsto \Phi^l \cdot f^{l-1} + h^l$, is exchanged for an arbitrary linear function $\phi^l : L_V(X^{l-1}) \to L_V(X^l)$.

Now we could formulate the definition of a neural network in a more generalized way:

Definition 11 (Feed forward neural network.)**.** A neural network is a function $\mathcal{N} : \mathcal{L}_{\mathbb{V}}(G/H^0) \to \mathcal{L}_{\mathbb{V}}(G/H^L)$, where G is a compact group, H a subgroup of G and $\mathbb{V}$ a vector space. $\mathcal{N}$ is a composition of functions $\phi^l : \mathcal{L}_{\mathbb{V}}(G/H^{l-1}) \to \mathcal{L}_{\mathbb{V}}(G/H^L)$ such that $\phi^l(f^{l-1})(g) = \sigma^l(\rho^l(f^{l-1})$ for some linear function ρ^l and a pointwise nonlinearity σ^l.

Since these networks are only defined for compact groups G, we will in the following always assume, when referring to a group G, that it is compact.

3.3.1 Generalize convolution

Since our interest lies in the properties of convolutional neural networks, we need to define what is meant by a convolutional layer in these new networks.

To begin with, the mathematical definition of convolution is the following:

Definition 12. Let G be a compact group and $\mathbb{V}$ a vector space. The convolution of two functions $f, g : G \to \mathbb{V}$ is $(f * g) : G \to \mathbb{V}$ where

$$(f * g)(u) = \begin{cases} \sum_{v \in G} f(uv^{-1})g(v) & \text{if G is finite or countable} \\ \int_{v \in G} f(uv^{-1})g(v)d\mu(g) & \text{if G is uncountable.} \end{cases}$$

If we look at how a convolutional neural network first was defined, we had that the linear operator in the network was of the form $\mathbf{x} \mapsto \mathbf{y}$ where $y_j = \sum_{i=1}^{k} x_{i+j-1}c_i$ for some vector $\mathbf{c} \in \mathbb{C}^k$, y_j is

only defined for $j \in \{1, 2 \ldots, n-k+1\}$. If we look at $\mathbf{x}, \mathbf{y}$ and $\mathbf{w}$ as functions f, g, h we get:

$$\begin{aligned} y_j = \sum_{i=1}^{k} x_{i+j-1} c_i \Leftrightarrow g(j) &= \sum_{i=1}^{k} f(i+j-1)h(i) && (\textit{let } l = 1-i) \\ &= \sum_{l=1-k}^{0} f(j-l)h(1-l) && (\textit{let } h'(x) = h(1-x)) \\ &= \sum_{l=1-k}^{0} f(j-l)h'(l). \end{aligned} \tag{5}$$

Since $h(i) = 0$ whenever $j \notin \{1, 2, \ldots, k\}$, we get $h'(i) = 0$ whenever $i \notin \{1-k, 2-n, \ldots, 0\}$. Hence the convolution between f and h' becomes:

$$\begin{aligned} (f * h')(j) &= \sum_{i \in \mathbb{Z}} f(j-i)h'(i) && h' = 0 \textit{ outside } \{1-k, \ldots 0\} \\ &= \sum_{i=1-k}^{0} f(j-i)h'(i). \end{aligned} \tag{6}$$

Comparing the function g in equation (5) and the convolution in equation (6), we see that for each j where $g(j)$ corresponds to an element y_j, it is the same as the convolution $(f * h')(j)$. Remember that $g(j) = y_j$ whenever $j \in \{1, \ldots, n-k+1\}$, but $g(j) = 0$ otherwise. The convolution $(f * h')(j)$ can however be nonzero for j outside this range. This means that the vector that corresponds to the function $f * h'$ will be longer than the vector $\mathbf{y}$ which we get as output from a so called convolutional layer. Thus the operation used in a convolutional layers is not exactly convolution in the mathematical sense. It is however close, since for every element in the vector y_j we have that it is the same as $(f * h')(j)$, so it is only at the edges of the vector we get a difference.

Thus the results we show about convolution will hold approximately for convolution in classical neural networks. Hence we ignore this issue and work from the view that the operator used in convolutional neural networks is the same as the operation defined as convolution in the mathematical context.

So far, we have defined convolution to be a operator $* : L_{\mathbb{V}}(G) \times L_{\mathbb{V}}(G) \to L_{\mathbb{V}}(G)$ for a group G. That is the input is two functions in the same space and the output is a function in the same space. Following definition 11, we want the linear operator in a layer to be a function from one space $\mathcal{L}_{\mathbb{V}}(G/H_l)$ to another space $\mathcal{L}_{\mathbb{V}}(G/H_{l+1})$. Thus we want to define convolution that allows us to go from one quotient space of G to another.

Whenever we have a function f on a quotient space G/H, there is a corresponding function on G. This function is called the *lifting* of f to G.

Definition 13. Let G be a group and H a subgroup of G. The *lifting* of $f : G/H \to \mathbb{V}$ to G is the function

$$f^{\uparrow G}(u) = f([u]_{G/H})$$

where $[u]_{G/H}$ is the element uH in G/H.

If $f : H\backslash G \to \mathbb{V}$ or $f : H_1\backslash G/H_2 \to \mathbb{V}$ the lifting of f to G is defined similarly. Using this lifting, convolution can be defined between two functions over different quotient spaces of the same group G.

Definition 14 (Generalized Convolution)**.** Assume that $\mathcal{X}_1, \mathcal{X}_2$ are *(right, left or double)* quotient spaces of a compact group G, $f \in \mathcal{L}_{\mathbb{V}}(\mathcal{X}_1), \Psi \in \mathcal{L}_{\mathbb{V}}(\mathcal{X}_2)$, then the convolution of f and Ψ is defined in the following way:

$$(f * \Psi)(u) = \sum_{v \in G} f^{\uparrow G}(uv^{-1}) \Psi^{\uparrow G}(v).$$

As usual, the summation is exchanged to an integral whenever G is an uncountable group. Since we want the linear operator to take us from functions in $\mathcal{L}_{\mathbb{V}}(G/H_l)$ to functions in $\mathcal{L}_{\mathbb{V}}(G/H_{l+1})$ in our convolutional layer, we want to know which sort of functions Ψ to choose from.

Lemma 1. If $f : G/H_1 \to \mathbb{V}, \Psi : H_1\backslash G/H_2 \to \mathbb{V}$ then

$$f * \Psi : G/H_2 \to \mathbb{V}.$$

The proof of this lemma can be found in [Kondor and Trivedi, 2018].

We are now ready to define a convolutional layer in a neural network:

Definition 15 (Convolutional layer)**.** A convolutional layer in a neural network is a layer which sends $f_{l-1} : G/H_{l-1} \to \mathbb{V}$ to $f_l : G/H_l \to \mathbb{V}$ where

$$f_l = \sigma_l(f_{l-1} * \Psi_l)$$

for some function $\Psi_l : H_{l-1}\backslash G/H_l \to \mathbb{V}$, and some activation function σ_l.

Definition 16. Convolutional Neural Network. Let G be a compact group and $\mathcal{N}$ an $L+1$ layer feed-forward neural network in which the i:th index set is G/H_i for some subgroup H_i of G. We say that N is a G-convolutional neural network (G-CNN), if each of the linear maps $\phi_1, \dots, \phi_L$ in $\mathcal{N}$ is a generalized convolution of the form $\phi_l(f_{l-1}) = f_{l-1} * \chi_l$, with some filter $\chi_l \in L_{V_{l-1}\times V_l}(H_{l-1}\backslash G/H_l)$.

3.4 Equivariance of convolutional neural networks

Now, we are interested in if the equivariance property of classical convolutional neural networks is preserved in these more general CNN's. To simplify notation, convolution will be written as a sum over the elements of G, thus assuming that G is finite. All of the results still holds as long as G is compact, the sums are in this case exchanged for an integral over the group.

Definition 17 (Equivariant neural network)**.** Let $\mathcal{N}$ be a feed-forward neural network, and G be a group that acts on each index set $\mathcal{X}_0, \dots, \mathcal{X}_L$. Let $\mathbb{T}^0, \dots, \mathbb{T}^L$ be the corresponding actions on $L_{V_0}(\mathcal{X}_0), \dots, L_{V_L}(\mathcal{X}_L)$ defined by $\mathbb{T}^l_g(f_l)(x) := f_l(g^{-1}(x)), g \in G$. We say that $\mathcal{N}$ is a G-equivariant feed-forward neural network if, when the inputs are transwormed $f^0 \mapsto \mathbb{T}^0_g(f_0)$ (for any $g \in G$), the activations of the other layers transform as $f_l \mapsto \mathbb{T}^l_g(f_l)$.

The first step to show that G-convolutional neural networks are guaranteed to be equivariant to actions of G, is to show the equivariance of one convolutional layer.

Lemma 2. Let $f \in \mathcal{L}_{\mathbb{V}}(X^1)$ and $\Psi \in \mathcal{L}_{\mathbb{V}}(X^2)$. Where X^i is (left/right or double) quotient spaces of a group G. Let $\{\mathbb{T}_g\}$ be the corresponding induced group actions. Then

$$\mathbb{T}_g f * \Psi = \mathbb{T}_g(f * \Psi).$$

Proof. We have that

$$(\mathbb{T}_g f) \uparrow^G (u) = (\mathbb{T}_g f)([u]_{X^1}) = f(g^{-1}[u]_{X^1}) = f([g^{-1}u]_{X^1}) = f \uparrow^G (g^{-1}u).$$

Using this equality, we get

$$\begin{aligned}(\mathbb{T}_g f * \Psi)(u) &= \sum_{v\in G} (\mathbb{T}_g f) \uparrow^G (uv^{-1})\Psi(v) \\ &= \sum_{v\in G} f \uparrow^G (g^{-1}uv^{-1})\Psi(v).\end{aligned}$$

This expression is the same as the convolution of f and Ψ evaluated at $g^{-1}u$, so we get

$$(\mathbb{T}_g f * \Psi)(u) = (f * \Psi)(g^{-1}u) = \mathbb{T}_g(f * \Psi)(u)$$

for all $u \in G$, as required. □

Lemma 3. If a layer ϕ in a neural network is convolutional, then $\phi(\mathbb{T}_g f) = \mathbb{T}_g \phi(f)$.

Proof. Since a convolutional layer is of the form $\phi(f) = \sigma(f * \Psi)$ where σ is a pointwise function, we have for all $u \in G$

$$\phi(\mathbb{T}_g f)(u) = \sigma(\mathbb{T}_g f * \Psi)(u) = \sigma((\mathbb{T}_g f * \Psi)(u)).$$

Since convolution is equivariant with respect to the first argument, this equals

$$\sigma(\mathbb{T}_g(f * \Psi)(u)) = \sigma((f * \Psi)(g^{-1}u)).$$

Using that σ is taken pointwise, the right hand side equals

$$\sigma(f * \Psi)(g^{-1}u) = \phi(f)(g^{-1}u) = \mathbb{T}_g \phi(f)(u).$$

Hence we have that $\phi(\mathbb{T}_g f)(u) = \mathbb{T}_g \phi(f)(u)$ for all $u \in G$, as required. □

Lemma 4. If ϕ^1 and ϕ^2 are both equivariant with respect to the action $\{\mathbb{T}_g\}$ of a group G, then $\phi^2 \circ \phi^1$ is also equivariant with respect to this action.

Proof.

$$(\phi^2 \circ \phi^1)(\mathbb{T}_g f) = \phi^2(\phi^2(\mathbb{T}_g f)) = \phi^2(\mathbb{T}_g \phi^1(f)) = \mathbb{T}_g \phi^2(\phi^1(f)) = \mathbb{T}_g(\phi^2 \circ \phi^1)(f).$$

□

Theorem 3. Let G be a compact group, $H_0, H_1, \ldots, H_L$ be subgroups and $\mathcal{N}$ a feed forward network with L layers in which the l:th layers is a mapping $\phi : \mathcal{L}_{\mathbb{V}}(G/H_{l-1}) \to \mathcal{L}_{\mathbb{V}}(G/H_l)$. Then $\mathcal{N}$ is an equivariant neural network if all layers are convolutional.

Proof. Using lemmas 3 and 4 and induction over l we get the desired result. □

4 Restricted Boltzmann Machines

One algorithm that uses neural networks is the so called restricted Boltzmann machine (RBM). It is used to model probability distribution over the input space, and is hence an example of unsupervised learning. Since we have generalized neural networks and shown how convolution guarantees equivariance, it is now interesting to see how these results transfer to the RBM. In this chapter we will present the classical RBM and two deep learning algorithms built by connecting RBM's. In chapter 5 we will transfer the results from chapter 3 to these learning algorithms.

4.1 Ising model

One way to model a distribution over vectors $\mathbf{x} = (x_1, \dots, x_n)$ of binary random variables, is by a graph with undirected edges. Each node, i, in the graph corresponds to a random variable x_i and each edge $\{i, j\}$ defines an interaction strength W_{ij} between the random variables x_i and x_j. We also have some constant b_i at each node that defines the strenght of the node itself. This graph represents that given an random vector $\mathbf{x}_i$, in which we let all the random variables be fixed, except the i'th one, we have that $p(x_i = 1|(x_j)_{j\neq i}) = \sum_j x_j W_{ij} + b_i$. Here we let $W_{ij} = 0$ if the edge $\{i, j\}$ is not in the graph.

Given such graph representing our random vector, the energy of a given vector is the function $E(\mathbf{x}) = -[\sum_{ij} x_i x_j W_{ij} + \sum_i b_i x_i]$. The probability of a vector $\mathbf{x}$ is given by $p(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})}$, where Z is a normalizing constant, making sure that p is a probability distribution. This way to model the random variables x_i is called an Ising model.

4.2 Restricted Boltzmann Machines

The RBM is an algorithm used to model the probability distribution over the input space. Why we are interested in this algorithm is because it is built up in a similar way as a neural network. Since we have proved that a convolutional neural network is equivariant to group actions, it is interesting to explore the properties of restricted Boltzmann machines when using convolution.

In a restricted Boltzmann machine, the Ising model is used to model the probability distribution over binary vectors $\mathbf{v} \in \{0, 1\}^n$. However, we do not use the Ising model directly on the vector $\mathbf{v} = (v_1, \dots, v_n)$. To capture deeper relations between the nodes in the input, we extend $\mathbf{v}$ with a so called hidden vector $\mathbf{h} = (h_1, \dots h_m) \in \{0, 1\}^m$. The weights between nodes in $\mathbf{v}$, and between nodes in $\mathbf{h}$ will be set to zero. Hence there will only be nonzero weights $w_{i,j}$ between nodes v_i and h_j for $i = 1, \dots, n$ and $j = 1, \dots, m$.

Remark. The word restricted in RBM refers to restricting the weights between nodes in the same layer being set to zero. There are Boltzmann machines that does not use this restriction [Montúfar, 2018].

The weights $w_{i,j}$ and the biases a_i, b_j are the parameters we want to optimize during training.

So in a restricted Boltzmann machine we have a layer of visible units $\mathbf{v}$, and one layer of hidden units $\mathbf{h}$. Let $\theta = \{w_{i,j}, a_i, b_j | i = 1, \dots, n, j = 1, \dots, m\}$. We define the energy of a given vector $\mathbf{x} = (\mathbf{v}, \mathbf{h})$ as

$$E_\theta(\mathbf{v}, \mathbf{h}) = -[\sum_{i,j} h_j v_i W_{ij} + \sum_i a_i v_i + \sum_j b_j h_j]$$

and the joint probability is the Boltzmann distribution, defined by this energy function [Hinton, 2012]:

$$p_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E_\theta(\mathbf{v},\mathbf{h})} = \frac{1}{Z} e^{[\sum_{i,j}, h_j v_i W_{ij} + \sum_i a_i v_i + \sum_j b_j h_j]}.$$

Summing over all possible hidden vectors $\mathbf{h}$ we get the probability distribution over the input space:

$$p_\theta(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{[\sum_{i,j} h_j v_i W_{i,j} + \sum_i a_i v_i + \sum_j b_j h_j]} \tag{7}$$

We also have the following:

$$p_\theta(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \qquad \text{where } p_\theta(h_j = 1|\mathbf{v}) = \sigma(\sum_i W_{i,j} v_i + a_j), \tag{8}$$

$$p_\theta(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) \qquad \text{where } p_\theta(v_i = 1|\mathbf{h}) = \sigma(\sum_j W_{i,j} h_j + b_i). \tag{9}$$

The function σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

Here we can se the similarity between Boltzmann machines and neural networks, the function $p_\theta(h_j = 1|\mathbf{v})$ in equation (8) is defined exactly as one layer in a neural network. The difference is that we do not use this vector as an output. Instead, what we are interested in is the probability distribution over the input space defined in equation (7).

The goal when training a Boltzmann machine is that given a set of input examples $\{\mathbf{v}_n\}_{n=1}^N$ we adjust the parameters θ so that the probability of drawing these samples given that they came from the distribution in equation (7) is maximized.

Let $\mathbf{v}$ be one example. To maximize the probability of drawing this vector, we can as well maximize the log-probability which is easier to calculate the derivative of. It can be shown that [Hinton, 2012]

$$\frac{\partial \log p_\theta(\mathbf{v})}{\partial w_{i,j}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}.$$

Where $\langle * \rangle_{data}$ denotes the expected value with respect to the data distribution and $\langle * \rangle_{model}$ is the expected value with respect to the model distribution.

$\langle v_i h_j \rangle_{data}$ can be estimated in the following way: First draw one of training examples randomly, then set h_j to 1 with probability $p_\theta(h_j = 1|\mathbf{v})$.

Unfortunately it is time consuming to obtain an estimation of$\langle v_i h_j \rangle_{model}$. This is solved by approximating this term with another term $\langle v_i h_j \rangle_{recon}$, obtained in the following way:

1. Set state of visible units to a random training example $\mathbf{v}$.
2. Compute states of hidden units $\mathbf{h}$, conditioned on $\mathbf{v}$ by equation (8).
3. Compute a reconstruction $\mathbf{v}'$ of the input $\mathbf{v}$, conditioned on the vector $\mathbf{h}$, acquired in step 2, with equation (9).
4. Now we estimate $\langle v_i h_j \rangle_{recon}$ as we did $\langle v_i h_j \rangle_{data}$ but use $\mathbf{v}'$ as the sampled vector instead of $\mathbf{v}$.

The updating rule for the parameters $w_{i,j}$ is then

$$\Delta w_{i,j} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}).$$

The update rule for the biases a_i, b_j works similarly.

Now $w_{i,j}$ is updated by $w_{i,j} = w_{i,j} + \Delta w_{i,j}$ and then repeat the process until we reach some stopping criteria. Using this updating rule is actually closer to minimizing another function than the log-likelihood, called the Kullback-Leibler divergence. This function measures the distance between two distributions. However, using this updating rule has shown to work quite well in practice. [Hinton, 2002]

4.3 Gaussian Bernoulli Restricted Boltzmann Machines

If the input we want to analyze is real valued instead of binary, then we have to use another distribution to model the input space. The corresponding algorithm is called a *Gaussian-Bernoulli RBM*. As before, we use a binary hidden layer $\mathbf{h} \in \{0,1\}^m$.

The energy function of such a model is

$$E_\theta(\mathbf{v},\mathbf{h}) = \sum_{i=1}^{n} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{n}\sum_{j=1}^{m} W_{i,j} h_j \frac{v_i}{\sigma_i} - \sum_{j=1}^{m} a_j h_j.$$

The probability distribution is still $p(\mathbf{v},\mathbf{h}) = \frac{1}{Z} e^{-E_\theta(\mathbf{v},\mathbf{h})}$ but with the above energy instead. The distribution over the input space is then

$$p_\theta(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})}. \tag{10}$$

The conditional distribution over each input unit is

$$p_\theta(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma} \exp - \frac{(x - b_i - \sigma_i \sum_{j=1}^{m} h_j W_{i,j})^2}{2\sigma_i^2}$$

and the conditional distribution over the hidden units is

$$p(h_j = 1|\mathbf{v}) = g(b_j + \sum_{i=1}^{n} W_{i,j} \frac{v_i}{\sigma_i}).$$

Even though the Gaussian Bernoulli RBM is relevant for our work, we will for simplicity restrict our focus in the rest of the the chapters to binary RBM's.

4.4 Deep Learning Based on Boltzmann Machines

Restricted Boltzmann machines is only built up of two layers, which limits its capacity to represent more complex probability distributions. In the same way we build deep neural networks by connecting many layers, we can connect restricted Boltzmann machines together to create deeper structures. This section will introduce two different ways to connect restricted boltzmnn machines to create such deeper structures. The first algorithm is called a deep Boltzmann machine, and the second is called deep belief network. These algorithms was presented in [Salakhutdinov, 2015] .

4.4.1 Deep Boltzmann machines

Recall that, in a RBM the input vector $\mathbf{v}$ is extended with a hidden vector $\mathbf{h}$. Then the distribution over vectors $\mathbf{x} = (\mathbf{v},\mathbf{h})$ is modeled with an Ising model. A straightforward way to introduce new hidden layers to the restricted Boltzmann machine would be to extend the vector $\mathbf{v}$ with not only one vector $\mathbf{h}$ of hidden variables, instead one could extend it with L vectors $\mathbf{h}^{(1)},\ldots,\mathbf{h}^{(L)}$. This is how we construct a deep Boltzmann Machine (DBM), with the restriction that all edges between units will have zero weight except when the units lies in adjacent layers. Thus we will only have weights $w_{i,j}^l$ between units $h_i^{(l-1)}$ and $h_j^{(l)}$. Given this construction, using the Ising model we get the energy

$$E(\mathbf{v},\mathbf{h}^{(1)},\ldots,\mathbf{h}^{(L)}) = -\mathbf{v}^T W^{(1)} \mathbf{h}^1 - \sum_{l=2}^{L} \mathbf{h}^{(l-1)T} W^{(l)} \mathbf{h}^{(l)}$$

and probability distribution over all layers.

$$p_\theta(\mathbf{v}, \mathbf{h}^{(1)}, \dots, \mathbf{h}^{(L)}) = \frac{1}{Z} e^{-E_\theta(\mathbf{v}, \mathbf{h}^{(1)}, \dots, \mathbf{h}^{(L)})}.$$

The probability of a certain unit being on given the previous and next layer is:

$$\begin{aligned} p_\theta(h_j^{(l)} = 1 | \mathbf{h}^{(l-1)}, \mathbf{h}^{(l+1)}) &= \sigma(\sum_i w_{i,j}^{(l)} h_i^{(l-1)} + \sum_m w_{j,m}^{(l+1)} h_m^{(l+1)}) \qquad (l = 1, \dots L-1) \\ p_\theta(v_i = 1 | \mathbf{h}^{(1)}) &= \sigma(\sum_j w_{i,j}^{(1)} h_j^{(1)}) \\ p_\theta(h^{(L)} = 1 | \mathbf{h}^{(L-1)}) &= \sigma(\sum_i w_{i,j}^{(L)} h_i^{(l-1)}). \end{aligned}$$

The probability of a given visible vector is

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}^{(1)}, \dots, \mathbf{h}^{(L)}} e^{-E(\mathbf{v}, \mathbf{h}^{(1)}, \dots, \mathbf{h}^{(L)})}.$$

There are different ways to train deep Boltzmann machines. Many of the algorithms are time-consuming and complicated, so developing a training algorithms is still a field of research [Desjardins et al., 2012].

4.4.2 Deep Belief Nets

Another deep learning algorithm based on a RBM is the deep belief network (DBN). A deep belief net is a combination of a restricted Boltzmann machine and a similar network, called a *deep sigmoid belief network*.

A sigmoid belief network is closely related to a Boltzmann machine. It uses a model that resembles the Ising model, but differs in the way that the edges in the graph are directed. Let $\mathbf{x} \in \{0, 1\}^N$ be the vector we want to model and $w_{i,j}$ be the interaction strength from x_j to x_i. In this new model we can model the probability over one unit conditional on the values of the units preceeding it [Neal, 1992]:

$$p_\theta(x_i = 1 | x_j : j < i) = \sigma(\sum_{j<i} w_{i,j} x_j + a_i), \tag{11}$$

$$p(x_1 = 1) = \sigma(a_1). \tag{12}$$

Here, as usual $\theta = \{w_{i,j}, a_i\}$ are the parameters defining the distribution.
Now, in a similar way as with the Boltzmann machine, if we want to model a vector $\mathbf{v}$ of visible units, we can extend it with a layer of hidden binary units $\mathbf{h}$. To get a sigmoid belief network the model described above is used on the vector $\mathbf{x} = (\mathbf{h}, \mathbf{v})$, where we let $w_{i,j} = 0$ whenever both units i and j lies in the visible layer, or both lies in the hidden layer. It is important that the hidden units have lower index than the visible units. In this setting equations (11) and (12) becomes:

$$\begin{aligned} p(v_i = 1 | \mathbf{h}) &= \sigma(\sum_j w_{i,j} h_j + b_i) \\ p(h_j = 1) &= \sigma(c_j). \end{aligned}$$

The probability of the vector $\mathbf{x} = (\mathbf{h}, \mathbf{v})$ is defined as

$$p(\mathbf{h}, \mathbf{v}) = p(x_1) \prod_i p(x_i | x_j : j < i) = \prod_i p(v_i | \mathbf{h}) \prod_j p(h_j) = p(\mathbf{v} | \mathbf{h}) p(\mathbf{h}).$$

We can stack multiple sigmoid belief nets to get a deep structure called a deep sigmoid belief net. Assume that we have L binary hidden layers $\{\mathbf{h}^{(l)}\}_{l=1}^{L}$. Let $\mathbf{v}{=}\mathbf{h}^{(0)}$, then we have the conditional probabilities

$$p(\mathbf{h}^{(l-1)}|\mathbf{h}^{(l)}) = \prod_i p(h_i^{(l-1)}|\mathbf{h}^{(l)}) \tag{13}$$

$$where\ p(\mathbf{h}_i^{(l-1)} = 1|\mathbf{h}^{(l)}) = \sigma(\sum_j w_{i,j}h_j^{(l)} + c_j^{(l)}) \tag{14}$$

The probability over the last layer is

$$p(\mathbf{h}^{(L)}) = \prod_i p(\mathbf{h}_i^{(L)})$$

$$where\ p(\mathbf{h}_i^{(L)} = 1) = \sigma(c_i^{(L)}).$$

The model of the deep sigmoid network is:

$$p(\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)}) = p(\mathbf{h}^{(L)}) \prod_{l=0}^{L-1} p(\mathbf{h}^{l}|\mathbf{h}^{l+1}).$$

A deep belief net is closely related to both deep sigmoid belief networks and Boltzmann machines. If we have L hidden layers, the first $L-1$ are connected as a deep sigmoid network, with directed edges. The layer $L-1$ and L are connected as a restricted Boltzmann machine, so the edges connecting the units in these layers go both ways. The probability distribution over all layers is

$$p(\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)}) = p(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)} \prod_{l=0}^{l-2} p(\mathbf{h}^{l}|\mathbf{h}^{(l+1)}).$$

The conditional probabilites is defined as in a sigmoid network and the probability $p_($ $\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)})$ is defined as in a restricted Boltzmann machine.

In the deep belief network, the training is done layerwise. Where we train the connections between layer $l-1$ and l as a restricted Boltzmann machine.

5 Convolutional Boltzmann machines

In previous chapters we have shown the benefits of using convolution in neural networks. Convolutional neural networks have been successful in practice, and there is a large amount of previous work exploring these networks and how to interpret convolution when we have a general group acting on the input. Most of this work is focused on supervised learning problems. In the following we want to explore what happens if we use convolutional connections in unsupervised learning algorithms. Since the Boltzmann machine is such an unsupervised algorithm, and it uses the same structure as neural networks, it would be interesting to study what happens when we restrict the connections between hidden and visible layers to be convolutional.

In this chapter we will introduce convolutional Boltzmann machines and explore their properties. As with neural networks, we first have to generalize the notation for Boltzmann machines, so that we can allow the input to be acted on by a general group.

5.1 Generalizing Boltzmann machines

In the following we want to define more general types of Boltzmann machines, allowing other types of inputs than vectors. This will be done using the generalization introduced in section 3. As far as we are aware of, this is the first attempt to generalize restricted Boltzmann machines to allow input acted on by any compact group G. For simplicity, we will focus on binary Boltzmann machines.

Using the notation introduced in chapter 3 we can see the input $\mathbf{v} \in \{0,1\}^n$ as a function $f^{(0)} \in L_{\{0,1\}}(\mathbb{Z}), f(i) = 0$ when $i \notin \{1,\dots,n\}$. In the more general definition of the Boltzman machine, we let the input be a function $f^{(0)} \in L_{\{0,1\}}(X^{(0)})$ where X is a homogeneous space of a group G. The hidden layer will be another function $f^{(1)} \in L_{\{0,1\}}(X^{(1)})$, where $X^{(1)}$ also is a homogeneous space of G. The energy function will be defined as

$$E_\theta(f^{(0)}, f^{(1)}) = f^{(1)} \cdot (f^{(0)} \cdot \Phi) + f^{(0)} \cdot \alpha + f^{(1)} \cdot \beta. \tag{15}$$

Where $\Phi \in L_{\mathbb{R}}(X^{(0)} \times X^{(1)}), \alpha \in L_{\mathbb{R}}(X^{(0)})$ and $\beta \in L_{\mathbb{R}}(X^{(1)})$ are the functions we want to find during training.

The distribution over the input space given the parameters $\theta = \{\Phi, \alpha, \beta\}$ is

$$p_\theta(f^{(0)}) = \frac{1}{Z} \sum_{f^{(1)} \in L_{\{0,1\}}(X^{(1)})} e^{-E_\theta(f^{(0)}, f^{(1)})} \tag{16}$$

with E_θ as in equation (15).

Definition 18 (Restricted Boltzmann Machine)**.** Let G be a group and $X^{(0)}$, $X^{(1)}$ homogeneous spaces of G. A restricted Boltzmann machine is a model of a distribution over the space $\{f : X^{(0)} \to \{0,1\}\}$. This model is defined as in equation (16).

5.2 Convolution in Boltzmann machines

As with neural networks, under certain restrictions of Φ, the linear operation $f \cdot \Phi$ can be seen as a convolution $f * \Psi$, defined as in chapter 3.3.1. The same restrictions can be done in a Boltzmann machine. Since the index sets are restricted to be homogeneous spaces of G, we can have $X^{(0)} \cong G/H^{(0)}, X^{(1)} \cong G/H^{(1)}$ for some subgroups $H^{(0)}$ and $H^{(1)}$ of G. The energy function of a Convolutional Boltzmann is then defined as:

$$E_\theta(f^{(0)}, f^{(1)}) = -f^{(1)} \cdot (f^{(0)} * \Psi). \tag{17}$$

where $\Psi \in L_{\mathbb{R}}(H^{(0)} \backslash G / H^{(1)})$ is the learnable parameter. When the learnable parameters in a Boltzmann machine are restricted to be of this form, then the model is called a *convolutional* Boltzmann Machine.

Definition 19. A convolutional restricted Boltzmann machine is a restricted Boltzmann machine in which the energy function is restricted to be of the form in equation (17).

Since in a convolutional RBM, θ contains less parameters than for an ordinary RBM, and these are restricted to be of a special form, we sometimes refer to this as θ being of convolutional form.

5.3 Properties of Convolutional Boltzmann machines

In neural networks, convolutional layers guarantee that the output of the network is equivariant to group actions on the input. In this section we will show a property for convolutional Boltzmann

machines which is called *invariance*. Invariance is a special case of equivariance.

Definition 20 (Invariance)**.** Let $X^{(1)}$ and $X^{(2)}$ be two sets acted on by a group G. A function $\phi : X^{(1)} \to X^{(2)}$ is called G-invariant, or invariant if the group is understood by context, if

$$\phi(\mathbb{T}_g x) = \phi(x) \text{ for all } g \in G \tag{18}$$

This means that a function that is G-invariant is not affected in any way by actions of G on the input.

For the Boltzmann machine there is not an output for each input, instead we end up with a probability distribution over the input space based on all examples. Thus if we want to explore properties about the Boltzmann machine, we have to look at the probability distribution it defines, that is $p_\theta : L_{\{0,1\}}(G/H^{(0)}) \to [0,1]$ defined as in equation (16). To talk about equivariance for such a probability distribution would not make sense, since G is arbitrary and there is no guarantee that it can even act on the output space $[0,1]$. However, we are still interested in exploring what happens if the input is acted on by the group G, given that θ is of convolutional form.

As it turns out the probability distribution defined by a convolutional Boltzmann machine is invariant to group actions:

Theorem 4. Let $X^{(0)}$ be a homogeneous space of a group G. Then a convolutional Boltzmann machine with input in $L_{\{0,1\}}(X^0)$ is G-invariant. That is

$$p_\theta(\mathbb{T}_g f^{(0)}) = p_\theta(f^{(0)}).$$

for any $g \in G$.

To prove this we first need two simple lemmas.

Lemma 5. If $f, f' \in L_{\mathbb{V}}(X)$ for some space X acted on by G and field $\mathbb{V}$, then $f \cdot \mathbb{T}_g f' = \mathbb{T}_{g^{-1}} f \cdot f'$.

Proof.

$$f \cdot \mathbb{T}_g f' = \sum_{u \in X} f(u) f'(g^{-1}u) \tag{19}$$

Since the sum is taken over all elements $u \in X$, it can as well be taken over $v = g^{-1}u \in X$. With this variable substitution, equation (19) becomes

$$\sum_{v \in X} f(gv) f'(v) = \mathbb{T}_{g^{-1}} f \cdot f'.$$

□

This lemma together with the equivariance of convolution shows that the energy function in a convolutional Boltzmann machine is invariant to group actions of both inputs:

Corollary 1. If θ is of G-convolutional form, then

$$E_\theta(\mathbb{T}_g f^{(0)}, \mathbb{T}_g f^{(1)}) = E_\theta(f^{(0)}, f^{(1)}) \text{ for all } g \in G.$$

Proof. When θ is of G-convolutional form, it holds that

$$E_\theta(\mathbb{T}_g f^{(0)}, \mathbb{T}_g f^{(1)}) = -\mathbb{T}_g f^{(1)} \cdot (\mathbb{T}_g f^{(0)} * \Psi). \tag{20}$$

Using lemma 5 and equivariance of convolution, equation (20) can be rewritten as $-\mathbb{T}_{g^{-1}}\mathbb{T}_g f^{(1)} \cdot (f^{(0)} * \Psi)$. Now group actions has the property that $\mathbb{T}_{g^{-1}}\mathbb{T}_g = Id$, where Id is the identity function. Thus

$$-\mathbb{T}_{g^{-1}}\mathbb{T}_g f^{(1)} \cdot (f^{(0)} * \Psi) = -f^{(1)} \cdot (f^{(0)} * \Psi) = E_\theta(f^{(0)}, f^{(1)}).$$

□

This corollary can be reformulated in the following way.

Corollary 2. If θ is of convolutional form, then

$$E_\theta(\mathbb{T}_g f^{(0)}, f^{(1)}) = E_\theta(f^{(0)}, \mathbb{T}_{g^{-1}} f^{(1)}) \tag{21}$$

Proof. This follows directly from corollary 1. □

Now we are ready to prove theorem 4.

Proof of theorem 4. We have that

$$p_\theta(\mathbb{T}_g f^{(0)}) = \frac{1}{Z} \sum_{f^{(1)} \in L_{\{0,1\}}(X^{(1)})} e^{-E_\theta(\mathbb{T}_g f^{(0)}, f^{(1)})} \tag{22}$$

Using corollary 2 together with a variable substitution $\mathbb{T}_{g^{-1}} f^{(1)} = h^{(1)}$, we can rewrite equation (22) as

$$\frac{1}{Z} \sum_{f^{(1)} \in L_{\{0,1\}}(X^{(1)})} e^{-E_\theta(f^{(0)}, \mathbb{T}_{g^{-1}} f^{(1)})} = \frac{1}{Z} \sum_{h^{(1)} \in L_{\{0,1\}}(X^{(1)})} e^{-E_\theta(f^{(0)}, h^{(1)})} = p_\theta(f^{(0)})$$

as desired. □

That the probability distribution defined by a convolutional Boltzmann machine is invariant to group actions means that the probability of drawing a certain example from this distribution is equal to the probability of drawing the same example, acted on by a group element from G. If we consider the case where we have a spherical image, this means that rotating the picture does not change the probability of it being drawn. In some applications of spherical images, the image represent a surrounding, with the viewer in the middle. In such a scenario, rotation invariance would mean that given a certain surrounding, the viewer turning around or looking up and down, does not change the probability of this surrounding. Given this perspective, there are certainly scenarios when the property of being invariant is beneficial.

In the upcoming sections we show that this property holds for the deeper structures based on Boltzmann machines presented in section 4.4.

5.4 Generalizing Deep Boltzmann machines and deep belief nets.

The invariance property for convolutional restricted Boltzmann machines can be shown to hold for the deeper structures built up from Boltzmann machines. This will be proved in the next section. First, we will show the probability distributions for deep Boltzmann machines and deep belief networks when using more general notation.

The input to both of these structures will be functions $f^{(0)} \in L_{\{0,1\}}(X^{(0)})$. There will be L hidden layers $\{f^{(l)}\}_{l=1}^{L}$ where $f^{(l)} \in L_{\{0,1\}}(X^{(l)})$. Each X^l is a homogeneous space of a group G. In both DBM's and DBN's the output will be a probability distribution over the input space $L_{\{0,1\}}(X^{(0)})$.

5.4.1 Deep Boltzmann Machine

The energy function of a Deep Boltzmann machine is

$$E_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)}) = -\sum_{l=1}^{L} f^{(l)} \cdot (f^{l-1} \cdot \Phi^{(l)}). \tag{23}$$

And the probability distribution over all layers are

$$p_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)}) = \frac{1}{Z} e^{-E_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)})}$$

with E_θ as in 23. The probability distribution over the input space is found by summing over all possible hidden functions

$$p_\theta(f^{(0)}) = \frac{1}{Z} \sum_{f^{(1)} \in X^{(1)}} \cdots \sum_{f^{(L)} \in X^{(L)}} e^{-E_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)})}. \tag{24}$$

Definition 21. Let G be a group and $X^{(0)}, X^{(1)}, \dots X^{(L)}$ be homogeneous spaces of G. A deep Boltzmann machine is a model of a distribution over the space $L_{\{0,1\}}(X^{(0)})$, defined as in equation (24).

Let $H^{(0)}, \dots, H^{(L)}$ be subgroups of G such that $X^{(l)} = G/H^{(l)}$. In a convolutional deep Boltzmann machine, each dot product $f^{(l-1)} \cdot \Phi$ in the energy function is exchanged to a convolution $f^{(l-1)} * \Psi$ with a function $\Psi \in L_{\mathbb{R}}(H^{(L-1)} \backslash G / H^{(l)})$.

5.4.2 Deep Belief Net

In a deep belief network with L hidden layers, we have the probability distribution

$$p_\theta(f^{(0)}, f^{(1)}, \dots f^{(L)}) = p(f^{(L-1)}, f^{(L)}) \prod_{l=0}^{L-2} p(f^{(l)} | f^{(l+1)}), \tag{25}$$

where $f^l \in G/H^l$. The probability $p(f^{(L-1)}, f^{(L)})$ is defined as in a restricted Boltzmann machine:

$$p_\theta(f^{(L-1)}, f^{(L)}) = \frac{1}{Z} e^{-E_\theta(f^{(L-1)}, f^{(L)})} \tag{26}$$

where E_θ is defined as in equation (15).

The conditional probabilities $p(f^{(l-1)} | f^{(l)})$ are the generalization of 13:

$$p(f^{(l-1)} | f^{(l)}) = \prod_{i \in G/H^{l-1}} p(f^{(l-1)}(i) | f^{(l)}) \tag{27}$$

$$\textit{where } p(f^{(l-1)}(i) = 1 | f^{(l)}) = \sigma((f^{(l)} \cdot \Phi^{(l)})(i) + \beta^{(l-1)}(i)).$$

Definition 22. Let $X^{(0)}, X^{(1)}, \dots, X^{(L)}$ be homogeneous spaces of some group G. A deep belief net is a model of a probability distribution over the space $L_{\{0,1\}}(X^{(0)}$ defined as

$$p_\theta(f^{(0)}) = \sum_{f^{(1)} \in L_{\{0,1\}}(X^{(1)})} \dots \sum_{f^{(L)} \in L_{\{0,1\}}(X^{(L)})} p_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)})$$

where $p_\theta(f^{(0)}, f^{(1)}, \dots, f^{(L)})$ is defined as in equation (25).

When θ is of convolutional form, we restrict all products $f^{(l)} \cdot \Phi^{(l)}$ to be a convolution $f^{(l)} * \Psi^{(l)}$. For $l < L$, $\Psi^{(l)} \in L_{\mathbb{R}}(H^{(l)} \backslash G / H^{(l-1)})$, and for the last layer we have $\Psi^{(L)} \in L_{\mathbb{R}}(H^{(l-1)} \backslash G / H^{(l)})$.

5.5 Properties of deep structures based on RBM's

5.5.1 Invariance of Convolutional Deep Boltzmann Machines

We will now show that the property of being invariant to group actions, which we showed for RBM's, extends to deep Boltzmann machines. This will be done by first showing that the energy function that defines the probability distribution for the DBM is invariant to having the same group element acting on all the inputs. The proof will follow the same steps as for the RBM.

Lemma 6. Let $X^{(0)}, \dots X^{(L)}$ be homogeneous spaces of a group G. Let B be a deep Boltzmann machine with L layers where the l'th layer lies in in $L_{\{0,1\}}(X^{(l)})$. If B is convolutional then the energy function it defines has the following property

$$E(\mathbb{T}_g f^{(0)}, \dots, \mathbb{T}_g f^{(L)}) = E(f^{(0)}, \dots f^{(L)}).$$

Proof. Corollary 1 tells us that

$$\mathbb{T}_g f^{(l-1)} \cdot (\mathbb{T}_g f^{(l)} * \Psi) = f^{(l-1)} \cdot (f^{(l)} * \Psi)$$

Hence

$$\begin{aligned} E(\mathbb{T}_g f^{(0)}, \dots, \mathbb{T}_g f^{(L)}) &= -\sum_{l=1}^{L} \mathbb{T}_g f^{(l-1)} \cdot (\mathbb{T}_g f^{(l)} * \Psi) \\ &= -\sum_{l=1}^{L} f^{(l-1)} \cdot (f^{l} * \Psi) \\ &= E(f^{(0)}, \dots, f^{(L)}). \end{aligned}$$

□

From this lemma, the following result one can easily prove the following corollary.

Corollary 3. The energy function for a convolutional deep Boltzmann machine has the following property:

$$E(\mathbb{T}_g f^{(0)}, f^{(1)}, \dots, f^{(L)}) = E(f^{(0)}, \mathbb{T}_{g^{-1}} f^{(1)}, \dots, \mathbb{T}_{g^{-1}} f^{(L)}).$$

Now, we are ready to prove that the probability distribution defined by a convolutional deep Boltzmann machine is invariant to group actions of the input.

Theorem 5. If all layers in a deep Boltzmann machine are convolutional, then

$$p_\theta(\mathbb{T}_g f^{(0)}) = p(f^{(0)}).$$

Proof. In a deep Boltzmann machine, p_θ is defined as

$$p_\theta(\mathbb{T}_g f^{(0)}) = \frac{1}{Z} \sum_{f^{(1)} \dots f^{(L)}} e^{-E_\theta(\mathbb{T}_g f^{(0)}, f^{(1)}, \dots, f^{(L)})} \tag{28}$$

Corollary 3 tells us that equation (28) can be written as

$$\frac{1}{Z} \sum_{f^{(1)} \dots f^{(L)}} e^{-E_\theta(f^{(0)}, \mathbb{T}_{g^{-1}} f^{(1)}, \dots, \mathbb{T}_{g^{-1}} f^{(L)})}. \tag{29}$$

Now let $h^{(l)} = \mathbb{T}_{g^{-1}} f^{(l)}$ for $l = 1, \dots, L$. Then we can sum over the functions $h^{(l)}$ instead, which yields the equality

$$p_\theta(\mathbb{T}_g f^{(0)}) = \frac{1}{Z} \sum_{h^{(1)} \dots h^{(L)}} e^{-E_\theta(f^{(0)}, h^{(1)}, \dots, h^{(L)})} = p_\theta(f^{(0)}) \tag{30}$$

as desired. □

5.5.2 Invariance of convolutional deep belief nets

So far we have shown that using convolutional layers guarantees both RBM's and DBM's to be invariant to group translations. Now we want to prove that DBN's holds the same property.

Theorem 6. If all layers in a deep belief network are convolutional, then $p(\mathbb{T}_g f^{(0)}) = p(f^{(0)})$.

This proof will need a bit more care than the previous ones for the Boltsmann machines, so we will first present a series of lemmas from which the theorem will follow easily.

Lemma 7. If θ is of convolutional form then the probability in equation (26) has the following property.

$$p_\theta(\mathbb{T}_g f^{(L-1)}, f^{(L)}) = p_\theta(f^{(L-1)}, \mathbb{T}_{g^{-1}} f^{(L)})$$

Proof. This probability is defined as

$$p_\theta(\mathbb{T}_g f^{(L-1)}, f^{(L)}) = \frac{1}{Z} e^{-E(\mathbb{T}_g f^{(L-1)}, f^{(L)})} \tag{31}$$

with the same energy as for a restricted Boltzmann machine. Corollary 2 tells us that since θ is of convolutional form, $E_\theta(\mathbb{T}_g f^{(L-1)}, f^{(L)}) = E_\theta(f^{(L-1)}, \mathbb{T}_{g^{-1}} f^{(L)})$. Hence we get can rewrite equation (31) as

$$\frac{1}{Z} e^{-E(f^{(L-1)}, \mathbb{T}_{g^{-1}} f^{(L)})} = p_\theta(f^{(L-1)}, \mathbb{T}_{g^{-1}} f^{(L)}). \tag{32}$$

□

Lemma 8. If θ is of convolutional form, then the probability distribution in equation (27) satisfies

$$p_\theta(\mathbb{T}_g f^{(l-1)} | f^{(l)}) = p_\theta(f^{(l-1)} | \mathbb{T}_{g^{-1}} f^{(l)}). \tag{33}$$

Proof. To be able to prove this, we have to be a bit more careful with notation than we have been so far. When we write $p(f)$ for a certain f, what is really mean is $p(\hat{f} = f)$ where $\hat{f}$ is a random variable. So the probability on the left hand side in equation (33) becomes $p_\theta(\hat{f^{l-1}} = \mathbb{T}_g f^{l-1} | f^{(l)})$. With this notation, we get

$$p_\theta(\widehat{f^{(l-1)}} = \mathbb{T}_g f^{(l-1)} | f^{(l)}) = \prod_i p(\widehat{f^{l-1}(i)} = \mathbb{T}_g f^{(l-1)}(i) | f^{(l)}) \tag{34}$$

Now if $f : X \to \{0, 1\}$, let $I_f = \{i \in : f(i) = 1\}$ and $O_f = \{i \in \chi : f(i) = 0\}$. Then $X = I_f \cup O_f$, with I_f and O_f disjoint. Thus we can divide the product in equation (34) into two products.

$$\begin{aligned}
&\prod_{i \in I_{\mathbb{T}_g f^{(l-1)}}} p(\widehat{f^{(l-1)}}(i) = 1 | f^{(l)}) \prod_{i \in O_{\mathbb{T}_g f^{(l-1)}}} (1 - p(\widehat{f^{(l-1)}}(i) = 1 | f^{(l)})) \\
&= \prod_{i \in I_{\mathbb{T}_g f^{(l-1)}}} \frac{e^{(f^{(l)} * \Psi)(i)}}{1 + e^{(f^{(l)} * \Psi)(i)}} \prod_{i \in O_{\mathbb{T}_g f^{(l-1)}}} \frac{1}{1 + e^{(f^{(l)} * \Psi)(i)}} \qquad (35)
\end{aligned}$$

Now, let $j = g^{-1}i$. Since $\mathbb{T}_g f(i) = 1 \iff f(g^{-1}i) = 1 \iff f(j) = 1$, it follows that

$$i \in I_{\mathbb{T}_g f} \iff j \in I_f.$$

Thus, we can rewrite equation (35) as

$$\prod_{j \in I_{f^{(l-1)}}} \frac{e^{(f^{(l)} * \Psi)(gj)}}{1 + e^{(f^{(l)} * \Psi)(gj)}} \prod_{j \in O_{f^{(l-1)}}} \frac{1}{1 + e^{(f^{(l)} * \Psi)(gj)}}.$$

The equivariance of convolution gives that this product equals

$$\begin{aligned}
&\prod_{j \in I_{f^{(l-1)}}} \frac{e^{(\mathbb{T}_{g^{-1}} f^{(l)} * \Psi)(j)}}{1 + e^{(\mathbb{T}_{g^{-1}} f^{(l)} * \Psi)(j)}} \prod_{j \in O_{f^{(l-1)}}} \frac{1}{1 + e^{(\mathbb{T}_{g^{-1}} f^{(l)} * \Psi)(j)}} \\
&= p_\theta(\widehat{f^{(l-1)}} = f^{(l-1)} | \mathbb{T}_{g^{-1}} f^{(l)})
\end{aligned}$$

as required. □

Corollary 4. If θ is convlutional then

$$\begin{aligned}
p(\mathbb{T}_g f^{(L-1)}, \mathbb{T}_g f^{(L)}) &= p(f^{(L-1)}, f^{(L)}), \qquad \textit{and} \\
p(\mathbb{T}_g f^{(l-1)} | \mathbb{T}_g f^{(l)}) &= p(f^{(l-1)} | f^{(l)}).
\end{aligned}$$

Proof. This follows directly from lemma 7 and 8. □

Lemma 9. If all layers in a deep belief network are convolutional, then

$$p(\mathbb{T}_g f^{(0)}, \mathbb{T}_g f^{(1)}, \dots \mathbb{T}_g f^{(L)}) = p(f^{(0)}, f^{(1)}, \dots, f^{(L)}).$$

Proof. We have that

$$p(\mathbb{T}_g f^{(0)}, \mathbb{T}_g f^{(1)}, \dots, \mathbb{T}_g f^{(L)}) = p(\mathbb{T}_g f^{(L-1)}, \mathbb{T}_g f^{(L)}) \prod_{l=0}^{L-2} p(\mathbb{T}_g f^{(l)} | \mathbb{T}_g f^{(l+1)}).$$

Using corollary 4, the right hand side equals

$$p(f^{(L-1)}, f^{(L)}) \prod_{l=0}^{(L-2)} p(f^{(l)} | f^{(l+1)}) = p(f^{(0)}, f^{(1)}, \dots, f^{(L)}).$$

□

Proof of theorem 6. Now the theorem follows easily from lemma 9:

$$\begin{aligned} p(\mathbb{T}_g f^{(0)}) &= \sum_{f^{(1)} \dots f^{(L)}} p(\mathbb{T}_g f^{(0)}, f^{(1)}, \dots, f^{(L)}) \\ &= \sum_{f^{(1)} \dots f^{(L)}} p(f^{(0)}, \mathbb{T}_{g^{-1}} f^{(1)}, \dots, \mathbb{T}_{g^{-1}} f^{(L)}) \qquad (h^{(l)} = \mathbb{T}_{g^{-1}} f^{(l)}) \\ &= \sum_{h^{(1)} \dots h^{(L)}} p(f^{(0)}, h^{(1)}, \dots, h^{(L)}) \\ &= p(f^{(0)}). \end{aligned}$$

□

6 Discussion

In chapter 5, we use the theory already developed for generalizing neural networks to generalize the restricted Boltzmann machine. These new restricted Boltzmann machines thus allow input that is not restricted to lie in an Euclidean space, but in any space that can be acted on by some compact group. Further, we have shown that the equivariance property for convolutional neural networks leads to invariance of the restricted Boltzmann machine, when using convolutional connections between the visible and hidden layer. We have also shown that connecting such Boltzmann machines into deeper structures such as the deep Boltzmann machines, or deep belief networks preserves the invariance property as long as the connectons between each layer is of convolutional form. However, we have only covered the case when the input is binary. As we described there exist Gaussian Bernoully restricted Boltzmann machines, which allows real-valued functions as input instead of binary valued. For further work, it would be interesting to develop a corresponding generalization for these machines. If this was accomplished one could explore if the property of invariance would be kept when using convolutional connections.

As we have seen even with ordinary neural networks, when implementing the theory one often have to make some approximations. For example, when we have a function over the sphere as input, we have to use a discrete version of the sphere. There is however no way to do this while keeping the symmetry of the sphere exactly. The loss of symmetry makes the equivariance property of convolution a bit less exact. Because of this the theory developed for invariant Boltzmann machine should be tested by implementing such a convolutional Boltzmann machine before we could draw any conclusions about how useful the theory actually is in practice.

References

[UR,] *Utformning av rapporter och kandidatarbetens skriftliga presentation för Civilingenjörsprogrammen vid Chalmers tekniska högskola*. 2008. Göteborg: Chalmers Tekniska Högskola.

[Cohen and Welling, 2016] Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA. PMLR.

[Cohen et al., 2018] Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. (2018). Spherical CNNs. In *International Conference on Learning Representations*.

[Coors et al., 2018] Coors, B., Condurache, A. P., and Geiger, A. (2018). Spherenet: Learning spherical representations for detection and classification in omnidirectional images. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 525–541, Cham. Springer International Publishing.

[Desjardins et al., 2012] Desjardins, G., Courville, A., and Bengio, Y. (2012). On training deep Boltzmann machines. *arXiv preprint arXiv:1203.4416*.

[Esteves et al., 2018] Esteves, C., Allen-Blanchette, C., Makadia, A., and Daniilidis, K. (2018). Learning so(3) equivariant representations with spherical cnns. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 54–70, Cham. Springer International Publishing.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.

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

[Hinton, 2012] Hinton, G. E. (2012). *A Practical Guide to Training Restricted Boltzmann Machines*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kondor et al., 2018] Kondor, R., Lin, Z., and Trivedi, S. (2018). Clebsch–gordan nets: a fully fourier space spherical convolutional neural network. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 10117–10126. Curran Associates, Inc.

[Kondor and Trivedi, 2018] Kondor, R. and Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the action of compact groups. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2747–2755, Stockholmsmässan, Stockholm Sweden. PMLR.

[Lee et al., 2009a] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009a). 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. Association for Computing Machinery.

[Lee et al., 2009b] Lee, H., Pham, P., Largman, Y., and Ng, A. Y. (2009b). Unsupervised feature learning for audio classification using convolutional deep belief networks. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in neural information processing systems*, pages 1096–1104. Curran Associates, Inc.

[Lei et al., 2014] Lei, J., Li, G., Tu, D., and Guo, Q. (2014). Convolutional restricted Boltzmann machines learning for robust visual tracking. *Neural Computing and Applications*, 25(6):1383–1391.

[Montúfar, 2018] Montúfar, G. (2018). Restricted Boltzmann machines: Introduction and review. In Ay, N., Gibilisco, P., and Matúš, F., editors, *Information Geometry and Its Applications*, volume 252 of *Springer Proceedings in Mathematics Statistics*, pages 75–115, Cham. Springer International Publishing.

[Neal, 1992] Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113.

[Norouzi et al., 2009] Norouzi, M., Ranjbar, M., and Mori, G. (2009). Stacks of convolutional restricted Boltzmann machines for shift-invariant feature learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2735–2742. IEEE.

[Salakhutdinov, 2015] Salakhutdinov, R. (2015). Learning deep generative models. *Annual Review of Statistics and Its Application*, 2(1):361–385.

[Wang, 2018] Wang, L. (2018). Three-dimensional convolutional restricted Boltzmann machine for human behavior recognition from RGB-d video. *EURASIP Journal on Image and Video Processing*, 120(2018).