What auto-encoders could learn from brains

Generation as feedback in unsupervised deep learning and inference

Gerben van den Broeke

Master's thesis

January 2016

Abstract

This thesis explores fundamental improvements in unsupervised deep learning algorithms. Taking a theoretical perspective on the purpose of unsupervised learning, and choosing learnt approximate inference in a jointly learnt directed generative model as the approach, the main question is how existing implementations of this approach, in particular auto-encoders, could be improved by simultaneously rethinking the way they learn and the way they perform inference.

In such network architectures, the availability of two opposing pathways, one for inference and one for generation, allows to exploit the symmetry between them and to let either provide feedback signals to the other. The signals can be used to determine helpful updates for the connection weights from only locally available information, removing the need for the conventional back-propagation path and mitigating the issues associated with it. Moreover, feedback loops can be added to the usual usual feed-forward network to improve inference itself. The reciprocal connectivity between regions in the brain's neocortex provides inspiration for how the iterative revision and verification of proposed interpretations could result in a fair approximation to optimal Bayesian inference.

While extracting and combining underlying ideas from research in deep learning and cortical functioning, this thesis walks through the concepts of generative models, approximate inference, local learning rules, target propagation, recirculation, lateral and biased competition, predictive coding, iterative and amortised inference, and other related topics, in an attempt to build up a complex of insights that could provide direction to future research in unsupervised deep learning methods.

Preface

My focus in the last year or so has been on how to enable computers to perceive and interpret their sensory input and how to learn these skills solely from passive observation. When starting to work on this master's thesis, it quickly became clear that my envisaged approach to machine learning deviated so substantially from the main-stream research, that satisfactorily working out the ideas to a concrete implementation to experiment with would not be feasible within the designated time-frame. In writing this thesis, my goal has therefore been to compile my thoughts into a (more or less) coherent essay, that would convey to the reader the insights that I have acquired in my studies and would provide pointers to the sources that brought me the inspiration.

Since master's theses are notorious for never being read by anybody but one's supervisor (if even that), I have often imagined my target audience to be my future self, seeking to again understand some forgotten insights after having spent some years focussing on more important problems. In the delusive hope of attracting other readers, I have tried to avoid needlessly technical jargon and focus more on intuitive explanations than mathematical derivations, to make the writing relatively accessible and pursue the mission of inciting insight.

A few words of disavowal may be appropriate. While my writing may frequently sound self-assured, I do not claim to fully understand all the concepts that I write about. I do know that writing about them made me understand them much better — trying to explain a topic often reveals the holes in one's understanding. However, some of the thoughts and insights may turn out to be flawed, irrelevant, or plainly wrong. Also, I do not claim any of the ideas to be novel. I have copied and combined many ideas from other people, and every thing I came up with has probably been around long before me. The value in this writing is not intended to be in the introduction of ideas, but in their compilation.

I conclude with some quick thanks, firstly to my supervisor Tapani Raiko, who gave me so much freedom to follow my own interests in unsupervised learning, that I started to call him my *unsupervisor*.

Besides him, I thank Harri Valpola, Antti Rasmus, Jyri Kivinen, Aapo Hyvärinen, and whomever else I forget to mention, for the valuable discussions, tips and inspiration that helped shape my thoughts.

Contents

Introduction

Deep learning is a popular field of research nowadays, and its recent achievements have pushed the boundaries of what computers are capable of when it comes to tasks involving vision, speech recognition, language translation, and other types of complex data processing. The interesting aspect is that the algorithms are not specifically written to perform those tasks, they are written to *learn* to perform those tasks. Learning from experience may be the natural way of acquiring behaviour for humans and other animals, but this has never been so for computers.

Perhaps these techniques do not yet realise the long-fostered dream of engineering general intelligence (whatever it exactly means), but it would already be a great achievement if we can get computers to mimic one ability of brain-based creatures: learning how to interpret data by discovering the statistical structure in the previously observed data. It is this challenge that will be focussed on in this thesis, where we rely on the assumption that this can be done in ways that are not specific to the type of data we are dealing with.

To learn to interpret data, most contemporary deep learning methods apply variations of the same recipe, back-propagation-powered supervised learning. During the learning phase, a desired output value is specified for every datum, and the network's connection strengths are modified to steer the output closer to that target value for future input with similar values. This thesis however seeks to advance another paradigm, that of unsupervised learning, which has recently not kept up with the achievements of supervised learning, but is widely expected to ultimately be more relevant:

> "we expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object."[1]

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015). Deep learning. *Nature*, 521(7553):436–444

Of course there is also quite some research in unsupervised learning, but it seems that popular approaches borrow strongly from the techniques developed for supervised learning, even though the problem setting is substantially different. Auto-encoders and the like, a prevalent unsupervised approach, basically apply a supervised method to unlabelled data. Although this should not necessarily be an issue, designing methods specifically for the unsupervised setting would likely lead to better results. It seems that in auto-encoders there is much potential for exploiting the symmetry between their inference and generation side, which could be working together more tightly.

My personal impression is that the modus operandi in our field is to develop algorithms that we do not fully understand ourselves, but that we can make work acceptably thanks to generic optimisation methods and loads of computational power and training data. Regarding the computational power, the recent breakthroughs of deep learning have been attributed as much to the application of massive computational (GPU) power as to the invention of better algorithms. For example, Jürgen Schmidhuber writes the following about a record-breaking result achieved with "good old on-line back-propagation"[2]:

> "Since BP was 3–5 decades old by then, and pattern deformations 2 decades, these results seemed to suggest that advances in exploiting modern computing hardware were more important than advances in algorithms."[3]

[2] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220

[3] Jürgen Schmidhuber (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117

My hunch is that much better results could be obtained with more sophisticated and less computation-hungry methods, but coming up with fundamental improvements is difficult and may require going back to the drawing board several times. It sometimes feels like most research in the current deep learning hype is hunting for short-term improvements in a small corner of the vast search space of possible methods. The expectable result of such a process is to end up in a local optimum.

Luckily, there are still plenty of ideas around for alternative deep learning approaches, and this thesis builds on several of them. There is a remarkable amount of theories of how the brain functions and learns that have found only scant echo in machine learning research. Similarly, there have been many inventions and experiments in machine learning, but many are largely ignored or forgotten about.

The plan in this thesis is to explore some less conventional deep learning approaches, draw inspiration from how the neocortex in the brain is speculated to work, and try to fit several of these ideas together. Although methods can be approached from many angles, our main perspective is that we start from auto-encoders and try to let the generation side provide feedback to the inference side. Rather than converging to a concrete algorithm, the intent is to lay out the lines of thought that lead to the outline of an unsupervised learning paradigm, and provide insights and guiding principles for future research.

Before jumping in, a quick overview of what is ahead. The starting point, chapter 1, presents the core concepts of deep unsupervised learning, and the role generative models have in this. Chapter 2 reviews some well-known algorithms that implement these concepts, and that will be built upon later. Basic brain anatomy and theories of how it processes information are covered in chapter 3. The subsequent two chapters will treat two different aspects of how unsupervised deep learning could be improved: In chapter 4, we look at alternatives to back-propagation, in particular at how the generation side and inference side of auto-encoder-like networks could directly provide learning signals for each other. In chapter 5, the main question is how the inference process could be improved by using generation in a feedback loop. Finally, the ideas of these two chapters are then combined in the synthesis.

1

Unsupervised learning — in concept

Unsupervised learning is the art of detecting predictable structure in data, and is an essential capability of intelligent systems, be they natural or artificial. It is this process that enables humans to perceive objects and sceneries instead of flickering coloured dots, and words and phrases instead of mere sound frequencies. It is the lack of this process that renders a computer with microphone and camera still effectively deaf-blind, and responding no more aptly after years of duty than on its first day. Ameliorating the absolute stupidity of computers is the driving motivation behind this thesis, and deep neural networks are the assumed tool for the job.

The fundamental principle of unsupervised learning is that by observing a stream of data, for example the images from a camera, it is possible to derive a model of the relations between the data variables (in this case the pixels). For each kind of data these relations are different, and we will stick with natural images as our running example. Images are not at all arbitrary pixel matrices, but contain many often recurring systematicities or 'patterns'. Even when given no prior knowledge about the type of data, a learning algorithm could quite easily discover that values of neighbouring pixels are strongly correlated and that edges tend to continue. Subsequently, the idea of deep learning is that the presences of such discovered patterns can be found to be related among each other, leading to the deduction of progressively more complex inter-variable relations or 'higher level' patterns: lines form certain shapes; eyes usually come in pairs; penguins belong on ice.

In a neural network algorithm, the learning happens through the adjustment of strengths of connections between simple functional units, similar to neurons in the brain[1]. A dedicated set of input units provides the sensory input, and triggers the others to settle to a new state of activation. Activations correspond to the detection of patterns, so the unit activations collectively represent the interpretation of an input datum. While the input units present just a matrix of pixels, the

[1] We will stick with this terminology: brains have neurons, algorithms have units.

activations of units higher up express the image as a composition of shapes, objects, sceneries and more abstract concepts. This capability to learn to interpret data to is what we wish to achieve, because it is crucial for intelligent behaviour.

1.1 Relation with supervised learning

The premise of unsupervised learning is that just observing the data itself is sufficient to learn to interpret data, and for example learn to recognise koalas and penguins in images simply from their occasional appearances. This contrasts it with supervised learning, arguably the more popular field nowadays, in which an algorithm would be taught to distinguish the koalas and penguins by passing it the correct label along with each picture. Although a practical method when desiring a particular input-output mapping, purely supervised learning requires lots of labelled training data and computation. Moreover, it results in a rather inflexible system that is optimised for a single, fixed task, and is unable to adapt to changes in the data during its application, when labels are not available.

Unsupervised learning may overcome these issues of supervised learning, since it requires no labels to learn from, and learns a generic abstraction that should be usable for different tasks. But even though an unsupervisedly trained network, assuming it is powerful enough, may internally have learnt to distinguish koalas and penguins, it does not provide this knowledge in a simple output value. Its interpretation forms a distributed representation, consisting of possibly thousands of variables whose individual meanings may be hard to understand or use. The network could still be useful for tasks such as spotting anomalies, estimating similarity, and perhaps filling up gaps in data with sensible values, but often we desire it to specialise at detecting and reporting particular patterns (e.g. pedestrian detection) or producing a particular output (e.g. a translated sentence).

Unsupervised learning is therefore frequently used in combination with other types of machine learning to steer its interpretations towards what the task at hand requires. It is used to aid supervised learning, either applied as 'pre-training' or fused together into a semi-supervised algorithm, making the supervised learning task easier and thereby reducing its need for large quantities of labelled data. Likewise, aiding reinforcement learning could reduce the number of actions required to learn fruitful behaviour.

1.2 *Learning representations*

In order to aid subsequent learning tasks, the primary task of an unsupervised learning algorithm is, at least as considered in this thesis, to transform input data into a more useful, 'high-level' representation — a task also called representation learning. Basically, usefulness is achieved when it is made easier to find relations between the input variables and either target values (in supervised learning) or actions and rewards (in reinforcement learning). For example, learning how to discriminate koalas and penguins becomes much easier when given variables representing properties like furriness and wingedness instead of the original pixel colour values.

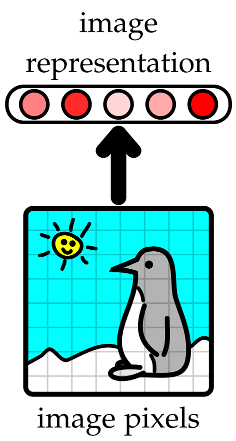

Figure 1.1: Pixels are interpreted to form a new representation of the image.

Although usefulness cannot be measured or even defined when the subsequent task is left unspecified, the general desire is that in the high-level representation the conceptual features implicitly present in a datum are disentangled:

> "the most robust approach to feature learning is to disentangle as many factors as possible, discarding as little information about the data as is practical."[2]

[2] Yoshua Bengio (2013). Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pages 1–37. Springer

For example, the colour of a boat can be considered a single feature, perhaps representable in a single variable, while in the datum it is spread out over many variables (pixels), which also contain information about other features, like the shape, size and position of the boat. Conversely, the representation should be largely invariant to conceptually small changes in the datum. For example, changing the boat's colour or moving it in the image should have only a small influence on its high-level representation.

The crucial question is how an algorithm can determine what are those features that are to be disentangled, when given only a bunch of apparently meaningless numeric vectors. The key is to reframe the problem in terms of statistics and mutual information, and try to formalise objectively measurable quantities that correspond as well as possible to the intuitive notion of conceptual features. As a first step, we can turn the above desire into two practical criteria:

1. The disentanglement turns the concepts into separate variables, and should thereby also reduce the redundancy found in the datum. A useful representation should have little redundancy, meaning that its variables should show little statistical dependence[3].

[3] Note that these criteria are very related to data compression, where reducing redundancy leads to shorter representations.

2. The representation should retain most information of the datum, and not cover only a few features while ignoring the rest. This implies that given only the representation, it should in theory be

possible to adequately reconstruct the datum.

Satisfying these criteria does not *guarantee* that useful representations are obtained. For example, desired high-level features might still be somewhat dependent (perhaps larger boats tend to be red more often), but the first criterion would disapprove this. These two criteria could be modified or added to, but for now they provide a simple and sensible basis to build upon[4]. Often we will assume one more criterion, which desires the representations to be sparse in activity, meaning that only a small portion of the variables has a non-zero value. The idea of using sparse representations matches well with the goal of disentangling concepts, since it can be assumed that each datum contains only a few from the plethora of possibly occurring concepts[5].

[4] See, for example, a theoretical treatise by Brendan van Rooyen et al. (2015): "*We desire a feature map that provides a compact representation of X*, that looses no information *about X*." (sic, their emphasis)

Brendan van Rooyen and Robert Williamson (2015). A theory of feature learning. *arXiv:1504.00083*

[5] See (Daniel Graham et al., 2006) for a theory of why the brain also uses sparse codes.

Daniel Graham and David Field (2006). Sparse coding in the neocortex. *Evolution of nervous systems*, 3:181–187

1.3 *Learning by generation*

The above criteria for good representations give a measure for the end goal, but little concrete guidance for how to pursue it. Many approaches have been developed that employ varying techniques to satisfy the above, or similar, criteria. A great many of them involve, in one way or another, a way to do the inverse of what we ultimately wish to achieve: they learn a way to reconstruct a datum from its representation. The intuition behind this is that in order to interpret data and represent it in terms of high-level features, one needs to understand how those high-level features manifest themselves in data. Succinctly put:

> "To recognize shapes, first learn to generate images"[6]

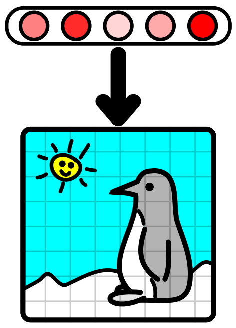

Figure 1.2: Reversing the arrow: learning to generate an image from its representation.

The previously still vague notion of 'interpreting' data now has gotten a clear task description, and that is to infer the causes that would have generated a datum when following the learnt model of generation[7]. In a sense, learning these rules is like searching for a better coordinate system for the data, a bit like how latitude and longitude provide a practical system to describe a spot on our three-dimensional globe.

[6] Geoffrey Hinton (2007b). To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547

[7] The causes in this generative model may or may not correspond to the real world causes of the datum, but the causal structure of our universe does provide a justification why directed generative models are useful for capturing the structure of our data.

The approaches investigated in this thesis learn a directed graphical generative model of the data. The idea of such models is to hypothesise the existence of latent variables, i.e. variables not observed in the data, that serve as the causes from which the data variables are assumed to have been generated. The values of the latent variables, in the network implemented as the activations of the 'hidden' units, influence the visible units (corresponding to the image pixels) through the downward connections, each contributing a particular pattern to the generated value (details follow next chapter). Learning proceeds by observing the data and finding a set of patterns that could have produced them as closely as possible, while assuming that statistical interdependencies between data variables are due to them having been influenced by the same latent variables, which themselves are ideally independent. The model thus learns patterns such that the latent variables satisfy both of the above criteria, low redundance and reconstructibility, meaning that they are closer to the conceptual features we wish to find and they can serve as our new representation.

Since a single step from latent variables to data space is limited to rather simple features, hierarchical generative models (see figure) deepen the idea by creating multiple layers of latent variables. The data is presented at the bottom, and each next layer tries to model the variables in the layer below itself. Through this sequence of several layers, patterns can interact and complex transformations can be performed, so that the higher layers can represent the data in terms of increasingly abstract features.

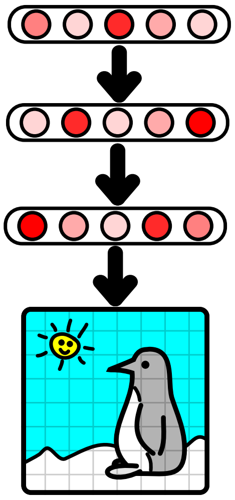
Figure 1.3: A hierarchical (deep) generative model.

> "Deep architectures lead to abstract representations because more abstract concepts can often be constructed in terms of less abstract ones."[8]

[8] Yoshua Bengio (2013)

The first latent variable layer could for example learn to generate short lines and edges in the pixel space, while the next layer learns to command the first layer where to place those lines and edges to form curves and shapes; and so on, up to penguins and koalas.

Although one could argue that a single layer model could also draw a koala, the marvel of abstraction is that high-layer variables do not just paint always the same pixel pattern onto the data canvas. Through the non-linear interactions, a 'koala variable' can cause a koala to be drawn in a pose and colour that is expressed by other variables. Abstraction is the power of deep networks, and reason enough to call other networks shallow. Note however, that the complexity created by the multiple layers also makes that learning in a deep network is much more difficult, and as we will now see, inference too.

1.4 *Inference*

The crucial remaining question is how exactly learning a generative model will help us with our goal of interpreting data. Having learnt how to generate items from high-level representations, in theory interpreting a datum should be possible by checking which representation would have generated it — in one word, by performing *inference*. Interpretation thus becomes the act of inferring the causes of the datum. Finding the best matching cause by trying all possible latent variable configurations is of course intractable, but ideally the computation for generation is invertible and given a datum its causes could easily be inferred. Unfortunately this is not the case, except in very simple models. Multiple layers and non-linearities in the generative model make that inference is much harder than generation, and besides it is even ill-defined since multiple representations could sometimes have generated the same result. Just learning a generative model is therefore not sufficient to learn to interpret. In fact it is not even practically possible, because, as we will see later (in 2.2.1), learning the generative model actually involves doing inference.

The common solution is to learn a separate inference model, that tries to approximately invert the generative model, and thus has its connections pointing in the opposite direction:

> "The role of the bottom-up connections is to enable the network to determine activations for the features in each layer that constitute a plausible explanation of how the network could have generated an observed sensory data-vector."[9]

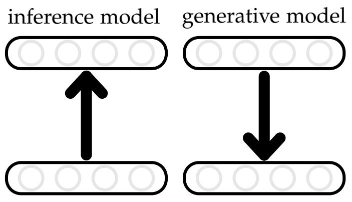

Figure 1.4: An inference model learns to approximate inference in the generative model.

[9] Geoffrey E Hinton (2007). Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434

The inference model is trained to produce representations that the generative model would turn back into the corresponding data. And vice versa, the generative model is trained to turn the representations produced by the inference model back into the corresponding data. Although it seems a chicken-and-egg problem[10] when gradually learning starting from a random initialisation, together these two models can attain a decent result[11]. This dual learning approach is the basic idea of auto-encoders, a family of algorithms that will be treated in the next chapter.

[10] Geoffrey Hinton (2007b) again phrases it aptly: "*So we have a chicken-and-egg problem: Given the generative weights we can learn the recognition weights and given the recognition weights we can learn the generative weights.*"

[11] This may however take many thousands of iterations, especially in deep networks; we will come back to this in section §4.6.

If we also have to learn an inference model anyway, we may ask if we still need a generative model at all. However, without the generative model we would be without guidance regarding which features to learn, so it seems we still do, even if only to ensure that useful representations are learnt. Learning the inference model could be seen as the primary goal, while the generative model acts as a constraint that ensures that the reconstructibility criterion is met. Moreover, a premise of this thesis is that the generative model can be more useful than just to guide learning of an inference model, so in chapter 5 we will entwine the two models and investigate how their combination can perform both tractable and powerful inference.

1.5 *Probabilistic versus deterministic models*

While up to now the models have been sketched quite generically, unsupervised learning methods can often be approached through either of two paradigms: deterministically, meaning exact values are computed for the variables, and probabilistically, where instead variables are assigned probability distributions of possible values to allow for uncertainty[12]. We will dive into the details later, but it may be good to highlight the conceptual differences. Methods are often classified as belonging to either paradigm, though the distinction is partly a matter of perspective, and a probabilistic method can be regarded as a probabilistic *interpretation* of an underlying deterministic method — though possibly one that includes a noise source. Probability theory gives a framework to think more abstractly about a problem, so by first designing a generative model in probabilistic terms, and then choosing how to turn it into a deterministic implementation that approximates ideal inference and learning, the resulting algorithm has an intuitive justification for what it does and can be more easily adjusted and compared to other algorithms.

[12] Connection strengths could be treated with uncertainty too, but we will not go there.

Probabilistic and deterministic approaches differ in how they regard generation. In a deterministic generation process, such as the decoder in a usual deep auto-encoder, each layer defines the exact value for the layer below. In contrast, in a probabilistic generative model, a value in one layer can for the layer below it produce any value from a whole distribution of values. Typically it would first compute a single expected value as would a deterministic model, and then deviate from it by adding a particular amount of noise, to end up with a value anywhere near the initial choice. When applying such variation in several layers, the distributions interact and make that a single representation at the top level can represent a complex distribution of possible values at the bottom. Each high-level representation can thus manifest itself in many ways, each manifestation having a different probability to occur.

Besides each representation generating a distribution of resulting values, a probabilistic generative model also defines a prior probability distribution that specifies how likely each representation is to occur itself. This solves the problem in inference that a datum could often have been generated from any of several representations, since inference can use the prior to assess how probable each possible cause is.

Insight: Priors add some sense to computers, providing a way to choose between technically correct (but possibly absurd) interpretations.

An important difference between deterministic and probabilistic generation is the amount of detail that has to be defined a top-level representation. In deterministic generation, the intermediate layers and bottom layer consist of fully dependent variables, with their values being uniquely determined by the top level representation[13]. The implication is that every detail in the generated item must already

[13] In the context of probabilistic models, such fully dependent variables are not even called variables at all.

be represented in the high-level representation, and conversely, an inference model cannot drop any details on the way up[14]. For example, even the change in the angle of a single straw in a picture of a meadow must correspond to a change in the high-level representation of that picture. This implication conflicts with our desire for a representation consisting of abstract features, and it wastes a lot of units to get all the details through the layers.

[14] Harri Valpola (2014). From neural PCA to deep unsupervised learning. *arXiv:1411.7783*

In a probabilistic model, each layer has freedom to pick from a set of values, which allows the top layers to ignore details that can be modelled in the lower layers. The top layers could command that grass should be drawn, but a lower layer can pick the angle for each straw. When performing inference, the lower layers can figure out and remember the angles of the straws, and report upwards that they see some grass. In a sense, the responsibility of representation is spread out over the layers, and the value of the top layer alone cannot be considered the full representation any more. The top level contains the high-level, abstract representation, while the lower layers contain the concrete details, together forming a *deep representation*.

Insight: Deep network $\neq$ deep representation. A deterministic deep network still produces flat representations if it is unable to leave details in lower layers.

2

Unsupervised learning — in algorithms

Many neural network based approaches to unsupervised learning have been explored in roughly the last half-century. Earlier experiments mainly aimed at testing hypotheses for the brain, while later the focus diverged and getting interesting results became more important than 'neural plausibility'. Besides uses of unsupervised learning for clustering and visualisation purposes, the unsupervised learning of representations (also dubbed representation or feature learning) gained popularity as a method for improving performance in supervised learning tasks. Probabilistic approaches led to generative models, also called density models, which could be seen as the extension of linear factor analysis, or even as supervised neural networks with the unspecified inputs[1]. This chapter will treat in more detail how the ideas from the previous chapter have been implemented in various concrete algorithms. Note that it is not a comprehensive overview of existing work, but rather covers a few related methods that are relevant for the remainder of this thesis; that is, those methods that employ some combination of an inference model and a generative model.

[1] David MacKay introduced density networks as such: "This is a neural network for which target outputs are provided, but the inputs are unspecified"

David MacKay (1995). Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, Section A*, 354(1):73–80

2.1 Neural networks

First we quickly recap the basics of neural network algorithms. As with perhaps any machine learning algorithm, there are two aspects to it:

1. A configurable model that performs a computation on a datum.
2. A learning algorithm that configures the model to perform the desired computation.

We will first look at the typical structure used in model architectures, and then at the typical way to configure (learn) them.

2.1.1 *Typical network architecture*

In the majority of contemporary approaches, a neural network computation is based on some variation of what is called a multi-layer perceptron network (MLP). Such a network involves layers of units, in which the input of a unit is the weighted sum (weighted by the learnt connection strength) of the activations units in the previous layer, plus a constant bias[2]. This implies that the layer as a whole performs an affine transformation on the column vector formed by the units previous layer, normally followed by an element-wise non-linear function. A deterministic inference network with the input layer $x = h_0$ at the bottom and an output h_L on top could be defined like this:

[2] The bias can also be negated to be regarded as a threshold, and for clarity it is often omitted in diagrams and formulas

$$h_l = \phi_l \left(W_l h_{l-1} + b_l \right) \qquad (1 < l \leq L)$$

In the most basic form (so basic it is often not even regarded as a neural net), there is only one layer and no non-linearity ($\phi(u) = u$). This is the case with both generation and inference in PCA and in ICA[3], and with generation in sparse coding (described below). A network's computation becomes more powerful when multiple layers with non-linearities are used. Commonly used non-linearities are sigmoidal functions and rectified linear units (ReLU), plotted in figure 2.1.

[3] Principal and independent component analysis are assumed familiar; for a good introduction to both, see (Aapo Hyvärinen et al., 2009). In short, principal component analysis rotates the data's coordinate system so that the first component (latent variable) represents the direction of largest variation in the data, and likewise each subsequent component explains as much of the data's remaining variance as possible. More interesting in our context is independent component analysis, which finds components that have as little dependence as possible among each other. It is however limited in this task by using only a single linear transformation.

Aapo Hyvärinen, Jarmo Hurri, and Patrick Hoyer (2009). *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.*, volume 39. Springer Science & Business Media

The patterns that units respond to or generate are defined through the weights W of the connections (and the bias b), and the learning process boils down to finding good values for those weights. Since the problem is too complex to analytically determine optimal weight values, the ubiquitous learning approach is to start from an initial (perhaps randomised) guess, and update the weights gradually while observing the data. The essential task of the learning algorithm is to determine the update directions that make the overall network perform a little bit better each time.

2.1.2 *Optimisation and objective functions*

Detached from the earlier neuroscience-based experiments, in which hypothesised models of neuron populations would be simulated with a chosen learning rule to see what patterns emerge, nowadays pretty much all machine learning algorithms are based on optimising the parameters of the model to make it better match a predefined objective. The parameters of the model, usually denoted by the vector θ, are essentially the connection weights $W_1 \cdots W_L$ and biases $b_1 \cdots b_L$ of the neural network, though sometimes other network properties are learnt too. The objective is a function $J(\theta, \mathcal{X})$, that evaluates how well the network performs with the parameter configuration θ on a data set $\mathcal{X}$. Often the objective is defined via a cost function C, and

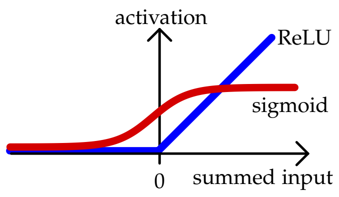

Figure 2.1: Shapes of ReLU and sigmoid activation functions.

$J(\theta, \mathcal{X}) = -C(\theta, \mathcal{X})$.

The optimisation is performed by some form of stochastic gradient ascent (or *descent* with regard to the cost), which basically amounts to iteratively computing the objective's gradient (its derivatives with respect to the parameters) on subsets of the data (mini-batches $\mathcal{X}_i$), and updating the model parameters a tiny step $\Delta\theta$ in the direction that locally satisfies the objective most:

$$\begin{aligned} \Delta\theta &= \eta \frac{\partial J(\theta, \mathcal{X}_i)}{\partial \theta} \\ \theta &\leftarrow \theta + \Delta\theta \end{aligned}$$

In practice, more sophisticated variations to this rule are used in order to speed up the optimisation, for example by dynamically adjusting the step size η, adding momentum, or using second order derivatives. If the model is a multi-layer neural network, computing the gradient involves back-propagating the partial derivatives through the layers, which will be a topic of chapter 4.

Whereas in supervised learning the objective is basically to have the network's output match the desired output (e.g. class label) for each of the training samples[4], in unsupervised learning it is not so obvious what to optimise for, and coming up with a good optimisation objective is perhaps the crux of unsupervised learning. In the deterministic case, the objective is normally defined in the form of a cost or 'energy' function that is to be minimised. Often the main term of such a cost function is the reconstruction error, which ensures that the network can generate the datum from the inferred representation. In probabilistic models, the objective function need not be manually defined but instead follows automatically from the form of the chosen probability distribution, as we will see shortly.

[4] Besides this main objective regularisation penalties are often added to prevent over-fitting the model on the training data

2.2 *Probabilistic models*

As discussed in section §1.5, we like to make abstractions around deterministic neural networks and instead think in terms of probability distributions, which often leads to the invention of approaches that we would not have come up with without this abstraction. Even though they may under the hood employ the same kinds of neural networks, probabilistic approaches may thus require quite a different way of thinking.

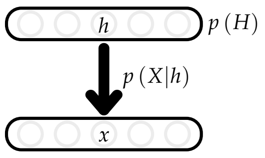

Figure 2.2: A directed probabilistic model defines a prior distribution $p(H)$ and a conditional distribution $p(X|h)$.

The conventional way to formalise probabilistic directed generative models is to define a parametrised distribution $p(X, H)$, with $p(x, h) = p(x|h)\,p(h)$[5]. The prior $p(h)$ is usually made factorisable so that each unit has a individual prior, for example a Gaussian (normal) or Laplace (sparse) distribution centered around zero. The

[5] For brevity, sloppy notation is used, e.g. $p(x|h)$ means $p_\theta(X = x|H = h)$.

conditional distribution $p\left(X|h\right)$ defines which values can be generated by a value h, which strongly depends on the setting of the parameters θ, in our case the neural network connection weights. Instead of the layer's value x, the neural network now generates the parameters for the conditional distribution. Most commonly, it generates the expected value μ of a Gaussian distribution:

$$p\left(X|h\right) = \mathcal{N}\left(\mu\left(h\right), \sigma^2 I\right)$$

To make the model hierarchical, technically speaking $p\left(h\right)$ is turned into a more complex distribution containing variables $h_1 \cdots h_L$, while $p\left(x|h\right)$ only depends on h_1. Explained more intuitively, each h_l is a layer of variables that can generate values for the layer below it, together forming a chain of conditional distributions:

$$p\left(x|h\right) p\left(h\right) = p(x|h_1)p(h_1|h_2)\cdots p(h_{L-1}|h_L)p(h_L)$$

Generating a single value can easily be done by ancestral sampling, first sampling a value for the top level h_L, then each level below it in turn, down to the data space at the bottom. Note that each conditional distribution $p\left(h_l|h_{l+1}\right)$ could in principle contain multiple deterministic layers that compute the distribution parameters, so a network can mix probabilistic and deterministic layers.

2.2.1 *Learning*

For probabilistic generative models, learning proceeds by trying to make our data distribution $p\left(x\right)$ as similar as possible to the data we have been given. Traditionally there have been two common approaches to quantify this similarity, which are actually equivalent. The first is to measure how likely it would have been to end up with the data set when drawing samples from the model distribution. The objective function could thus be the 'data likelihood' of a data set $\mathcal{X}$:

$$J\left(\theta, \mathcal{X}\right) = p(\mathcal{X}) = \prod_{x \in \mathcal{X}} p\left(x\right)$$

In practice, to make computation simpler and turn the product into a sum, the logarithm of this value is used instead. This is effectively equivalent because the logarithm is a monotonic function, making that the gradient will still point to the same direction, albeit with a different magnitude. The usual objective is thus the data log-likelihood:

$$J\left(\theta, \mathcal{X}\right) = \log p(\mathcal{X}) = \sum_{x \in \mathcal{X}} \log p\left(x\right) = \mathop{\mathbb{E}}_{x \sim q} \log p\left(x\right)$$

The right-most expression is obtained by considering the data as samples from a data distribution, named $q(X)$, which is itself unknown. Viewing the data as having originated from a probability distribution justifies the use of subsets (mini-batches) of the data to approximate the expectation, instead of summing over the whole data set. Also, it brings up the second approach, which is to make the model distribution $p(X)$ as similar as possible to $q(X)$ by trying to minimise the Kullback-Leibler divergence between them, a dissimilarity measure for probability distributions:

$$KL(q||p) = \mathop{\mathbb{E}}_{x\sim q} \log \frac{q(x)}{p(x)} = \mathop{\mathbb{E}}_{x\sim q} \log q(x) - \mathop{\mathbb{E}}_{x\sim q} \log p(x) = H(q) - \mathop{\mathbb{E}}_{x\sim q} \log p\left(x\right)$$

Because the data entropy $H(q)$ is constant and thus of no influence in the gradient calculation, setting $J\left(\theta, \mathcal{X}\right) = -KL\left(q||p\right)$ is equivalent to log-likelihood learning.

In either approach, the problem of choosing the objective function is reduced to defining the probability distribution. However, one big problem remains, because since our models define the distribution based on the latent variables, computing the marginal $p\left(x\right) = \sum_h p\left(x|h\right) p(h)$ (or its gradient with respect to θ) for even a single datum x is usually intractable, as it would require summing over all possible values of h. To avoid the full summation, the idea of the expectation–maximisation (EM) algorithm[6] can be applied: for each datum, it is estimated which setting of the latent variables would probably have caused it, and for some picked value(s) of h the probability $p\left(h, x\right)$ is increased by updating the parameters. Estimating a probable value however means that we need to sample from $p\left(h|x\right)$[7], and it is for this reason that inference is required to learn in a generative model[8].

[6] Arthur Dempster, Nan Laird, and Donald Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38

[7] Ideally we would compute the whole distribution, but sampling should give an unbiased estimate to approximate real EM.

[8] As was hinted in 1.4

2.2.2 *Inference*

Inference in a probabilistic generative model would ideally correspond to the calculation of $p\left(H|x\right)$, the probability distribution of representations given a datum x, the calculation for which is derivable through Bayes' rule:

$$p\left(h|x\right) = \frac{p\left(x|h\right) p\left(h\right)}{p\left(x\right)} = \frac{p\left(x|h\right) p\left(h\right)}{\sum_h p\left(x|h\right) p\left(h\right)}$$

Because of the involved summation over all representations, this is usually intractable. Luckily, it often suffices to either be able to draw samples from this distribution, or to compute only the single most probable representation. As these require only relative probabilities, the $p\left(x\right)$ can be dropped from the equation above. The single most probable interpretation, or maximum a posteriori (MAP) inference, thus becomes:

$$\hat{h} = \arg\max_{\mathrm{h}} p\left(h|x\right) = \arg\max_{\mathrm{h}} \frac{p\left(x|h\right) p\left(h\right)}{p\left(x\right)} = \arg\max_{\mathrm{h}} p\left(x|h\right) p\left(h\right)$$

Or, described in words, we search for the representation $\hat{h}$ that would be most likely to both occur itself and generate datum x. Unfortunately, searching $\hat{h}$ through this optimisation process is still too hard in any but the simplest models. As mentioned in 1.4, many methods avoid trying to compute $p(H|x)$ altogether, and instead go for a *variational* approach by trying to approximate $p(H|x)$ with an inference model, which is a separate neural network aligned in the opposing direction that learns a probability distribution $q(H|x)$[9]. This is the idea behind for example the variational auto-encoder (VAE), treated further below, and to some extent the idea can be found behind auto-encoders in general.

[9] Note that the same letter q is used again on purpose, and the data distribution $q(X)$ can sensibly be considered the prior of the inference model.

An important implication is that because the approximate inference model $q(H|x)$ is also used to replace the intractable $p(H|x)$ when learning the generative model, any imperfection of the approximation may partly be compensated for by the generative model. If the inference model produces wrong values of h given some x, the generative model will automatically adapt by learning to generate that x from that h, thereby effectuating that the inference model is more accurate again. The downside of this adaptation is that the generative model will be suboptimal and cannot become better than the approximate inference allows it to.

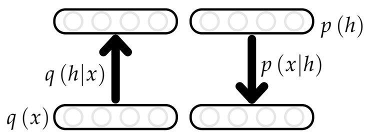

Figure 2.3: An inference model $q(h|x)$, with the prior $q(x)$ being the data distribution, should approximate $p(h|x)$.

Insight: q restrains p. Learning of a generative model is restrained by the quality of the inference model in use.

2.3 *Auto-encoders*

In an auto-encoder, a generative model is sided by an inference model, which provides the values h from which the generative model then learns to generate the original value x. Although we have just covered these ideas in a probabilistic framework, most auto-encoders lack a probabilistic interpretation, and words like inference and generation are replaced by more the technical terms encoding and decoding. Nevertheless, at least intuitively we can regard the decoder as a generative process and the encoder as an approximation to inference. Further down we will look at the more recent variational auto-encoder, which introduces a method that makes the connection between deterministic auto-encoders and probabilistic generative models.

2.3.1 *Basic auto-encoder*

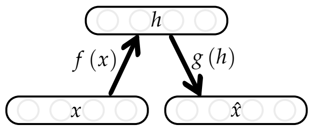

Figure 2.4: Simple sketch of an auto-encoder. The functions f and g could also contain multiple layers.

An auto-encoder learns a matching pair of functions: an encoder f, which transforms a given data sample (a vector $x \in \mathbb{R}^{d_0}$) into its new representation ($h_L \in \mathbb{R}^{d_L}$), and a decoder g, which turns such a representation h_L into a reconstruction $\hat{x} \in \mathbb{R}^{d_0}$. Both functions are implemented as neural networks. In the classical auto-encoder both consist of a single affine transformation, in the encoder possibly followed by a sigmoidal activation function. When either side has multiple layers (and non-linearities) the thing is dubbed a deep auto-encoder.

The encoder and decoder are trained together to ideally become each other's inverses for data samples. Put differently, for any x from the data, we desire $g(f(x)) = x$. This is pursued by choosing an objective function that penalises any deviations from this equality. Taking squared vector distance to measure the error in the reconstruction of the input, the basic auto-encoder cost function is:

$$C_{error} = \sum_{x \in \mathcal{X}_i} \|x - g(f(x))\|^2$$

Seen in another way, supervised learning is being performed on the MLP formed by the combination $g \cdot f$, while the target value is simply the input value itself. Commonly the decoder and encoder are given same (but flipped) shape in terms of depth and layer sizes. The weights of the encoder and decoder can be tied to be each other's transposes, which often speeds up initial learning but limits their ability to fine-tune.

Some care needs to be taken to make the auto-encoder learn useful representations of the data. Because only the criterion of reconstructibility is taken into account in the above cost function, it could end up with for example an identity mapping ($f(x) = x$, $g(h) = h$), which satisfies the cost function, but not our actual goal. The original way to overcome such trivial solutions is to simply make h of smaller dimensionality than x ($d_L < d_0$), so that it needs to learn how to compress and decompress the data, thereby reducing redundancy. However, this limitation of representation dimension is often undesirable, as it may be useful to have representations with a much larger dimensionality than the input data in order to disentangle many concepts.

To tackle the problem of ending up with trivial or absurd models, many variations to the basic auto-encoder have been developed, each effecting some form of regularisation.

2.3.2 *Denoising auto-encoder*

One approach at auto-encoder regularisation that is worth a quick mention is the principle of the denoising auto-encoder[10]. The method is like the standard, deterministic (deep) auto-encoder, except that

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

during training, the input data is partially corrupted before being passed into the encoder. The target value is uncorrupted, so the auto-encoder needs to learn how to turn corrupted data back into the original data. Thereby it learns not a function that reconstructs observed data as well as possible, but one that changes its input it towards values that are more probable in the data set. In other words, it ideally learns to compute derivative of the data distribution[11]:

[11] Guillaume Alain and Yoshua Bengio (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593

$$g\left(f\left(x\right)\right) = x + \frac{\partial q\left(x\right)}{\partial x}$$

Because the auto-encoder tries to be insensitive to noise in the input, it has to learn to exploit structure in the data to extract the information from noisy signals, and create representations that are minimally influenced by the noise. This last concept is also the idea behind contractive auto-encoders, which instead of adding noise explicitly calculate the derivative $\frac{\partial f(x)}{\partial x}$ of the representation to the input and add a penalty to the cost function to minimise it.

2.3.3 *Sparse auto-encoder*

Another approach to auto-encoder regularisation is to put extra constraints on the representation. In sparse auto-encoders, the variables in the representation are penalised when taking too many non-zero values, thereby encouraging the auto-encoder to find representations that have many values at or close to zero. The trick is implemented by adding a sparsity penalty to the cost function, for example using the absolute values of the hidden unit activations:[12]

[12] I use the notation $h[i]$ to mean the i^{th} element of h, because subscripts already indicate layer numbers.

$$C_{\text{total}} = C_{\text{error}} + C_{\text{sparse}} = C_{\text{error}} + \lambda \sum_i |h_L[i]|$$

Here λ is a hyperparameter[13] used to trade off reconstruction accuracy and representation sparsity. Because of the regularisation constraint, a sparse auto-encoder can have as many or more hidden units than input units without ending up learning a trivial solution.

[13] A hyperparameter is a parameter that is not updated by the learning algorithm, and often tweaked manually.

A point to note is that not all representations necessarily become sparse. Especially for data outside the training set, the encoder can equally happily produce dense representations, because only during training the sparsity is encouraged. To not confuse the decoder, it may be beneficial to enforce the sparsity of representations when using the trained network, by reducing the activity of representations that exceed some threshold[14]. More rigorously, sparsity could be enforced by the network also during training, replacing the need for a training penalty[15]. This idea leads to a broader insight:

[14] Kyunghyun Cho (2013). Simple sparsification improves sparse denoising autoencoders in denoising highly corrupted images. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 432–440

[15] *k*-Sparse auto-encoders do this, though there it actually proved better to enforce sparsity *only* during training

Alireza Makhzani and Brendan Frey (2013). k-sparse autoencoders. *arXiv:1312.5663*

Insight: Encouragement versus enforcement. Regularisation penalties can be substituted by architectural measures, which may help to improve training and/or to generalise better afterwards.

2.4 *Sparse coding*

Sparse coding[16] is a technique that, similar to sparse auto-encoders, learns to create sparse representations. Different from sparse auto-encoders, it only defines the generative side, or decoder, which is in this case simply a linear transformation by a matrix A[17]. Although originally defined in a purely deterministic way, the method appeared to be interpretable as a probabilistic model. Because a good understanding of the model is helpful for grasping the methods covered later in this thesis, we cover both explanations and show the equivalence.

[16] Bruno Olshausen et al. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609

[17] The letter A rather than W is conventional here.

2.4.1 *Deterministic definition*

Lacking an explicit encoder, the representation of a data vector x is implicitly defined as the vector h that is both sparse and accurately generates x when h is passed through the decoder:

$$h = f(x) = \arg\min_{h^\star} \left\{ \|x - Ah^\star\|^2 + \lambda \sum_i |h^\star[i]| \right\} \tag{2.1}$$

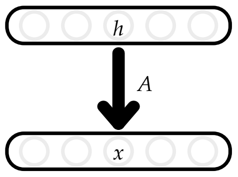

Figure 2.5: Sparse coding defines only generation explicitly.

Again, there is a trade-off choice between accuracy and sparsity. Lacking an encoder, the inference is performed by an optimisation algorithm, that searches for the best value h given the current matrix A and an input x. Instead of gradient-based optimisation as normally used in parameter learning, orthogonal matching pursuit is often used, which exploits the expected sparsity by starting from an all-zero representation and one by one activating latent variables that give the biggest step towards the objective[18]. The generative model is learnt by, after having inferred a value h, updating the matrix A towards exactly the same objective of minimising $\|x - Ah\|^2 + \lambda \sum_i |h[i]|$, while regarding h constant.

[18] Kyunghyun Cho (2014). *Foundations and Advances in Deep Learning*. PhD thesis, Aalto University

Although the iterative inference process can be significantly more expensive than having an explicit encoder network, it performs more powerful inference as it ensures that h generates x.[19]

[19] In section §5.5.1, we will treat this in depth.

2.4.2 *Probabilistic interpretation*

As was already mentioned, sparse coding can be interpreted probabilistically, and then appears to perform MAP inference on a probabilistic generative model. This model has a Laplacian prior distribution on the hidden layer h, in which values near zero are much more likely than larger values.

$$\begin{aligned} p\left(H\right) &= \text{Laplace}\left(\mu=0, b=\frac{1}{\lambda}\right) \\ p\left(h\right) &= \prod_i \frac{\lambda}{2} e^{-\lambda|h[i]|} \end{aligned}$$

The conditional distribution for generating a value x from a chosen h is a multivariate Gaussian around the computed expected value $\mu\left(h\right) = Ah$, with each variable being independent from its peers and having variance $\sigma^2 = 1/2$.

$$\begin{aligned} p\left(X|h\right) &= \mathcal{N}\left(\mu = Ah, \Sigma = \frac{1}{2}I\right) \\ p\left(x|h\right) &= \prod_i \frac{1}{\sqrt{\pi}} e^{-(x-Ah)[i]^2} \end{aligned}$$

Deriving MAP inference in this model shows the equivalence with the original definition in equation 2.1:

$$\begin{aligned} \hat{h} &= \arg\max_h p\left(h|x\right) \\ &= \arg\max_h p\left(x|h\right) p\left(h\right) \\ &= \arg\max_h \left\{\log p\left(x|h\right) + \log p\left(h\right)\right\} \qquad (2.2) \\ &= \arg\max_h \left\{\sum_i -\left(x - Ah\right)[i]^2 + \sum_i -\lambda\left|h[i]\right| + constants\right\} \\ &= \arg\min_h \left\{\left\|x - Ah\right\|^2 + \lambda \sum_i \left|h[i]\right|\right\} \end{aligned}$$

Learning in this probabilistic generative model should ideally be done by tweaking the parameters (just the matrix A in this case) to maximise the data log-likelihood $\mathbb{E}_{x\sim q} \log p\left(x\right) = \mathbb{E}_{x\sim q} \log\left(\sum_h p\left(x,h\right)\right)$. The sparse coding learning procedure instead optimises $\mathbb{E}_{x\sim q} \log p\left(x, \hat{h}\right)$, and can be regarded as a lower-bound approximation that uses the MAP value of h rather than all h or samples from its conditional distribution $p\left(h|x\right)$.

Having interpreted sparse coding probabilistically, we may view the sparse auto-encoder as nearly the same model, with the difference that it employs one more approximation, by replacing the MAP inference process with a parametrised inference model that is learnt alongside the generative model. Note however, that because inference is used during learning, replacing the inference procedure also influences the generative model that is learnt[20]. An interesting hybrid between the

[20] See insight "q restrains p", §2.2.2.

auto-encoder and sparse coding is predictive sparse decomposition, in which an encoder is used to provide a starting point for (and thereby speed up) the optimisation process of the inference[21].

[21] Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun (2010). Fast inference in sparse coding algorithms with applications to object recognition. *arXiv:1010.3467*

2.5 *Variational auto-encoders*

The variational auto-encoder (VAE)[22] makes a clear connection between the various kinds of deterministic auto-encoders of section §2.3 and the variational methods of probabilistic generative models of section §2.2, the latter of which involved using an inference model $q(h|x)$ that should approximate the intractable $p(h|x)$ to learn a generative model $p(x|h)\,p(h)$. Whereas with the deterministic auto-encoders we had to use quite some imagination to interpret them as approximations to generative models and inference, the VAE is built with a probabilistic foundation, from which a practical implementation is derived that is quite similar to an auto-encoder.

[22] Diederik Kingma and Max Welling (2013). Auto-encoding variational bayes. *arXiv:1312.6114*

The VAE learns by maximising, for each datum x, the not the log-likelihood $\log \mathrm{p}(x)$ directly, but a related objective $\mathcal{L}$:

$$\begin{aligned}\mathcal{L} &= \mathop{\mathbb{E}}_{h\sim q(h|x)} \{-\log q(h|x) + \log p(h,x)\} && (2.3)\\ &= -D_{KL}(q(H|x)\,\|\,p(H)) + \mathop{\mathbb{E}}_{h\sim q(H|x)} \log p(x|h) && (2.4)\end{aligned}$$

This objective is a lower bound to the data log-likelihood: $\log p(x) \geq \mathcal{L}$. More precisely, the discrepancy between the two is the Kullback-Leibler divergence between $q(h|x)$ and $p(h|x)$:[23]

$$\log p(x) = D_{KL}(q(H|x)\,\|\,p(H|x)) + \mathcal{L}$$

[23] Diederik Kingma et al. 2013, eq. 1

So as long as the inference model q adequately approximates inference in the generative model p, $\mathcal{L}$ is a fair substitute for the intractable log-likelihood $\log p(x)$.

To estimate the expectation in 2.3 or 2.4, the expected value over $q(H|x)$ is approximated by sampling some values of h from $q(H|x)$, and in fact even a single sample may suffice, which we will assume here[24]. This gives the approximation $\tilde{\mathcal{L}}$:[25]

$$\tilde{\mathcal{L}} = -D_{KL}(q(H|x)\,\|\,p(H)) + \log p(x|h)$$

[24] Diederik Kingma et al. (2013) found that "*the number samples L per data point can be set to 1 as long as the mini-batch size M was large enough*". Furthermore, the learning may be improved by taking several samples and weighting the samples by their quality based on $p(x,h)$ (Yuri Burda et al., 2015).

[25] Diederik Kingma et al. 2013, eq. 7

The first term can be considered a regularisation that tries to make the inference model's posterior match the generative model's prior. The second term attempts to increase the probability that the inferred h would generate the datum x. Although this term would seem to only affect the weights of the generative model (because h is a fixed sample), the interesting trick that makes the method very similar to deterministic auto-encoders is to reparametrise the inference model $q(H|x)$ so that the distribution is expressed as a deterministic function

$g(\epsilon, x)$ (including the neural network) of a fixed noise distribution $p(\epsilon)$[26]. This way sampling $h \sim q(H|x)$ can be done by first sampling $\epsilon \sim p(\epsilon)$ and then computing $h = g(\epsilon, x)$ to transform the noise sample to become a sample of $q(H|x)$. Now the objective $\tilde{\mathcal{L}}$ can be maximised not only by tweaking the generative model turn h into x, but also by tweaking the inference model (by tweaking the mapping g) to produce a value h that would more likely generate x. This is very much the learning principle of an auto-encoder, since both the inference model (encoder) and generative model (decoder) weights are updated simultaneously and by the same objective of reconstructing the input.

[26] We use the paper's notation of g and $p(\epsilon)$ here, do not confuse the letters with their other usages.

To illustrate the relatedness with a simple example, consider the case where $q(h|x) = \mathcal{N}\left(\mu_q(x), \operatorname{diag}\left(\sigma_q(x)^2\right)\right)$, i.e. a factorisable Gaussian distribution that computes it distribution parameters $\mu_q(x)$ and $\sigma_q(x)$ with perhaps an MLP. Similarly $p(x|h)$ is a Gaussian with $\mu_p(h)$, but we fix its variance to $\Sigma = \sigma_p^2 I$. We actually need not define the prior $p(h)$ now, as it would only influence how the representations are regularised. The inference model q can be reparametrised by setting the noise distribution $p(\epsilon)$ to be a standard Gaussian, and $g(\epsilon, x) = \mu_q(x) + \sigma_q(x) \odot \epsilon$ [27]. Now the 'reconstruction' term of the objective $\tilde{\mathcal{L}}$ becomes:

[27] The $\odot$ means element-wise multiplication

$$\begin{aligned} \log p(x|h) &\propto \frac{\left(x - \mu_p(h)\right)^2}{\sigma_p^2} \\ &= \frac{\left(x - \mu_p\left(\mu_q(x) + \sigma_q(x) \odot \epsilon\right)\right)^2}{\sigma_p^2} \end{aligned}$$

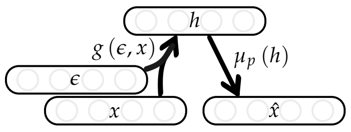

Figure 2.6: The VAE is like an auto-encoder with noise mixed into the encoder.

This part of the learning objective clearly resembles the squared-error reconstruction cost of a deterministic auto-encoder, where μ_p would be the decoder and μ_q the encoder, except that it adds a learnable and data-dependent amount of noise to the representation h; it could reasonably be considered a denoising auto-encoder that adds the noise to the representation instead of to the input. This example shows a very simple case, but a nice property of the VAE is that because it provides a probabilistic derivation, the optimisation objectives automatically follow after having defined the probability distributions.

3

Unsupervised learning — in brains

In the way the wings of birds have inspired people to invent flying machines, the brain is the source of inspiration for crafting intelligent machines. It is safe to claim that had brains not existed, the idea that intelligence is possible would not even have come up. However, aeroplanes do not flap their wings even though birds do, and likewise, artificial intelligence need not copy the way brains work. Nevertheless, after more than half a century of fervent tinkering, neither classical, rule-based AI nor connectionistic approaches (neural networks) have achieved the expected progress towards basic intelligence, so it seems that a lot may still be gained from investigating the principles applied by the brain, as it does some things right.

This chapter will cover some basics of brain anatomy, focussing on theories about how brains appear to implement unsupervised learning. In particular we focus on the use of generation or prediction in the neocortex, the principles of which will form the basis of the ideas covered in the remainder of this thesis.

3.1 The neocortex

The mammalian neocortex, often just called cortex, is the main source of inspiration for many deep learning enthusiasts. Compared to other brain parts, it is considered a more recent invention of evolution, and with its relatively large size in primates and especially humans (constituting nearly 80% of our brain[1]) it has been credited with being the engine of higher intelligence. It receives input from the senses via the thalamus, controls voluntary motor activity, and encompasses many of the complex processes that happen in between. Although the cortex continuously interacts with other brain parts and studying it in isolation may prove unavailing, it may be a good starting point to look at its anatomy.

[1] Henry Markram, Maria Toledo-Rodriguez, Yun Wang, Anirudh Gupta, Gilad Silberberg, and Caizhi Wu (2004). Interneurons of the neocortical inhibitory system. *Nature Reviews Neuroscience*, 5(10):793–807

Physically, the cortex consists of the outer sheet of both halves of the cerebrum.

The sheet is about 2 to 3 millimeters thick, containing in the order of 100 000 neurons per square millimeter[2]. Radially the neurons are arranged in layers[3] with different properties and connectivity. Most layers contain many lateral connections between the neurons in the same layer, and neurons have their dendrites (the tree of input collectors) and axons (the output tentacle) sprout up and down to connect to neurons in other layers. Usually, following the numbering by Korbinian Brodmann dating from 1909[4], six layers are distinguished:

- Layer 1 contains hardly any neuron bodies, but seems to host a networking party for axons and dendrites from the other layers.
- Layer 2 and 3 are rather similar and in many analyses taken together. They connect intensively laterally in patchy patterns, and their activation is quite sparse[5].
- Layer 4, called the granular layer, receives input from the thalamus or another cortical region, and unidirectionally relays to layer 3[6].
- Layer 5 seems a sort of output layer. It engages in motor control, sending outputs down to for example the spinal cord, as well as to the thalamus.
- Layer 6 is not intensively integrated locally, but mostly involved in communication with other cortical areas, as well as forming loops with the thalamus.

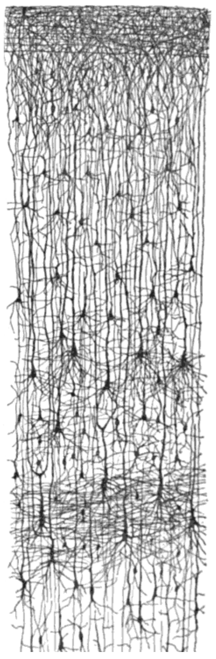

Figure 3.1: A Golgi-stained cut-through of the cortex of an infant, with approximate indication of layer locations. Adapted from Santiago Ramón y Cajal (1899).

[2] David Mumford (1991). On the computational architecture of the neocortex: I. The role of the thalamo-cortical loop. *Biological cybernetics*, 65(2):135–145

[3] Not to be confused with layers in artificial neural networks.

[4] Wikipedia (2015). Brodmann area — Wikipedia, the free encyclopedia

[5] Henry Markram et al. (2004)

[6] Stewart Shipp (2007). Structure and function of the cerebral cortex. *Current Biology*, 17(12):R443–R449

Disregarding minor deviations in some cortical areas, this layer structure appears all over the cortex. The column (through all six layers) of neurons located at roughly the same spot on the sheet are closely related in function, so rather than thinking about individual neurons, it seems more appropriate to regard a column of neurons as the primary computational unit, a so-called canonical microcircuit. This is not unlike how digital electronics can better be analysed in terms of logic gates (the micro-circuit) than transistors, although the boundaries and function of the grouping are in the neural case not agreed upon [7]. Columns with related processing functions (perhaps detecting similar features) are often located closer together and are heavily interconnected, forming fuzzy, cooperating assemblies[8].

Besides the described pyramidal or primary neurons, about 20%–30% of neurons is made up of various forms of interneurons[9], which connect only locally and most of which are inhibitory, meaning their activation discourages activation in the neurons they project

[7] Jonathan Horton and Daniel Adams (2005). The cortical column: a structure without a function. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):837–862

[8] Kenneth Harris and Thomas Mrsic-Flogel (2013). Cortical connectivity and sensory coding. *Nature*, 503(7474):51–58

[9] Henry Markram et al. (2004)

to (analogous to negative weights). Some types of interneurons connect unspecifically to any surrounding neurons, thereby apparently measuring and controlling average activity, creating sparsity through competition. As nearby columns commonly have correlated activity, such inhibition could help to decorrelate signals[10].

[10] Thomas Cleland and Christiane Linster (2012). On-center/inhibitory-surround decorrelation via intraglomerular inhibition in the olfactory bulb glomerular layer. *Frontiers in integrative neuroscience*, 6

On a larger scale, the cortical sheet can be segmented into several dozen regions or areas, having on average about 100 million neurons[11] that collectively process a multitude of related signals. The regions are richly interconnected, with each other as well as with subcortical brain parts such as the thalamus. Especially in regions topologically close to the senses, the connectivity between regions follows a hierarchical structure, which can perhaps best be studied by taking an example.

[11] David Mumford (1992). On the computational architecture of the neocortex: II. The role of cortico-cortical loops. *Biological cybernetics*, 66(3):241–251

3.2 *The visual cortex hierarchy*

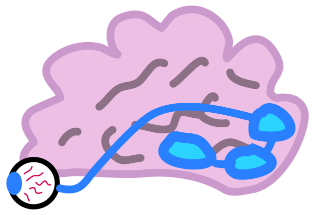
Figure 3.2: Rough location of regions of the visual cortex.

By far the best studied group of regions is the visual cortex, located in the back of the head (see sketch), and comprising roughly 32 regions in primates[12]. The primary visual cortex region, shortly V1, receives signals coming from the eyes (after some preprocessing outside the cortex), and its columns appear to be sensitive to the presence of features like edges and dots in different locations of the visual field[13]. The region is strongly connected with an adjacent region, V2, which receives the signals from V1, responds to more complex features, and in turn connects to subsequent regions. Each region in fact connects to many others — roughly a third of possible pairs of regions are connected[14] — and the topology elicits a richly connected hierarchical structure. The hierarchy processes and combines inputs from the senses in a gradual fashion, with the neurons in 'higher' regions (topologically further away from the senses) dealing with input from a larger receptive field and reacting to more complex features than the lower ones; a discovery that has provided substantial inspiration for the development of deep learning methods (where confusingly the analogue of a region is called a layer).

[12] Daniel Felleman and David Van Essen (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47

[13] David Hubel and Torsten Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106

[14] Daniel Felleman et al. (1991)

The connectivity between subsequent regions is reciprocal but asymmetric, with the ascending and descending pathways each connecting from and to different cortical layers. Although there are many exceptions to every discovered systematicity, there appear to be general rules or at least tendencies in the connectivity, and studying these typical connection patterns may give valuable clues about the computation being performed.

In each region, layer 4 appears to serve as the main entrance for ascending signals, which originate from layer 3 in the preceding region. In V1, having no predecessor, layer 4 receives visual signals from the retina, via the lateral geniculate nucleus (LGN) inside the thalamus. Inside the region, layer 4 passes signals to layer 3, thus paving a two-step path for ascending through a region[15]. In addition to connections within the cortex, subcortical loops also contribute to the ascending pathway. Outputs from layer 5 drive higher order relays in the pulvinar thalamus, which in turn connect mainly to layer 3 of the subsequent region[16,17].

[15] Stewart Shipp (2007)

[16] ibid.

[17] S. Murray Sherman and R. W. Guillery (2002). The role of the thalamus in the flow of information to the cortex. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 357(1428):1695–1708

Descending connections mainly originate from layer 6 and 5, to terminate in layer 1 and 6 in the lower region. Between regions close to each other in the hierarchy, the general rules are adhered to less strictly[18], and in particular extra descending paths connect from layer 2/3 to 1 and 6[19]. Remarkably, taken together the descending connections outnumber the ascending connections by an order of magnitude. Their influence is more often modulatory[20], meaning that they do not trigger any activity, but rather modify how neurons respond to driving signals, e.g. amplifying or inhibiting their activity.

[18] Stewart Shipp (2007)

[19] David Mumford (1992)

[20] Daniel Felleman et al. (1991)

3.3 *Unification theories*

The hierarchical connectivity of cortical regions appears not only in visual processing areas, but is similar for every sensory modality, and to some degree also continues into the higher-level, non-sensory regions. Seeing pretty much the same structure of regions, layers and columns of neurons repeat all over the cortical sheet, it seems that the cortex uses a single type of computation for tasks as diverse as vision, audition and language.

The physical similarity in structure can however not suffice as evidence, because of course the exact connectivity determines the performed computation. By analogy, different digital circuits can look similarly homogeneous, with straight rows of transistors forming standard cells, but perform completely different computations. A stronger support for the universality of the cortex is its neuroplasticity, the ability of cortical areas to adopt different functions. For example, primary auditory cortex (A1) can learn to process vision and becomes somewhat like V1 when the nerves from the eyes are rewired to replace those from the ears[21]. To a certain degree, the cortical areas thus appear to be configured by the signals they receive, and learn to process whichever kind of input is passed to them.

[21] Jitendra Sharma, Alessandra Angelucci, and Mriganka Sur (2000). Induction of visual orientation modules in auditory cortex. *Nature*, 404(6780):841–847

This property of the cortex matches exactly with what we are trying to achieve with unsupervised learning: given a stream of data, the machine should reconfigure itself and learn how to make sense out of that kind of data. The neocortex can thus be said to perform unsupervised learning, and the most intriguing question is then what wizardly computation the cortex performs in its regions and column micro-circuits. Unfortunately, despite a long-standing interest, quite little is understood about the computation it implements. Single neurons have been studied, and the functional areas have been mapped roughly, but the nature of its computation remains largely a mystery.

Ideally, we would come up with a high-level description of a computational design, which the neural structures we observe could be implementing. Like with machine learning algorithms, the task of cortex could be divided into two processes happening at different time scales: firstly the way in which the cortex processes given input, and secondly the way it reconfigures itself to learn how to process future input. Preferably, the design we come up with would have justifiable statistical or information-theoretic interpretation, covering both the inference and the learning aspect, and giving an explanation regarding why the particular computation is performed.

Of course, the task of cortex is much more varied than the learning of abstract representations we discussed earlier, not the least because it also involves actions like motor control and decision making. This may complicate the picture when we are mostly interested in the more passive functionality, but hopefully it does not prevent us from drawing inspiration from the neocortex for our current purposes. Even better, study of the cortex may ultimately provide hints for how to weave action taking and reinforcement learning into our algorithms.

3.4 Generative brains

Perhaps the most counter-intuitive finding in brain research is the massive amount of feedback connections, making that each pair of regions is connected reciprocally. This fact is at odds with the intuitive idea that signals should mainly flow forward, from the senses to the higher levels. Several theories propose that the descending pathway in one way or another implements a generative process, so that the brain learns not only to interpret sensory input, but also to construct possible sensory experiences from higher-level ideas. Conveniently, such an explanation would also provide a clue about the origins of dreams, imagination and hallucinations.

Unsupervised learning in the cortex could thus involve learning a hierarchical generative model, following the principles discussed in the first chapter. It would however be different from the machine learning algorithms, like those described in chapter 2, whose optimisation procedures, to begin with, seem not to match learning in the cortex. To adapt the synaptic strengths without access to a global objective's gradient, perhaps the descending pathway plays a direct role in learning of the ascending path, and vice versa. The application of this idea in machine learning will be explored in the next chapter.

More interestingly still, there are many signs that the cortex applies generation more extensively than just to facilitate learning of an inference model, suggesting rather that generation is an intrinsic part of inference. We will now review some influential theories that gave rise to and support this hypothesis, to look at applications of this idea in machine learning further on, in chapter 5.

3.4.1 Predictive brain hypotheses

A whole branch of theories hypothesise that the brain has an active role in processing its input, so that perception is largely an act of generation and prediction, rather than merely transforming incoming signals in a feed-forward manner. This school of thought got reasonable attention and development in the last few decades, although historical credit is often given to Hermann von Helmholtz, who already in the 1860s realised that perception is a result of an observer's expectations as well as the sensory input[22]. Although a controversial idea at that time, the role of prediction in the brain gained significant research attention later on.[23]

[22] Hermann von Helmholtz (1867). *Handbuch der physiologischen Optik*. Voss

[23] Note that besides foretelling future input, in this context prediction/expectation also includes 'spatial' prediction: expecting variables to have certain values based on values of other variables.

One contemporary researcher active in this theme is Stephen Grossberg, who has argued that a feedback system is required to explain observed psychological phenomena such as the overshadowing effect in conditioning[24,25]. In his adaptive resonance theory, sensory signals flow forward to invoke an initial interpretation in the higher level, which then produces a template (learnt from previous experiences) of the activity it expects in the level below. The lower level of cells receives this template, and emits a special signal if its values did not match the expectation, upon which the active higher level's active cells would be muted to allow an alternative interpretation to emerge. This process thereby performs error correction, and the essence of the design is that by using learnt feedback expectancies, coding errors (wrong interpretations) can be detected and corrected without any individual cell knowing that one occurred.

[24] Stephen Grossberg (1980). How does a brain build a cognitive code. *Psychological Review*, pages 1–51

[25] The overshadowing effect appears when a subject is conditioned to expect e.g. a shock when hearing a tone. When subsequently the tone is consistently accompanied by a flash of light, the subject will afterwards not fear the flash when it is given separately. If the initial conditioning of tone$\rightarrow$shock would have been omitted, presenting either a flash or tone would elicit a response of fear, suggesting that the brain compares its input to its expectations. See (Stephen Grossberg, 1980, sec. 2)

In 1992, David Mumford published an influential theory on the roles of the cortical pathways[26]. Like with Stephen Grossberg's model, a higher area would send its expectation template down to the lower area. The lower area now attempts to reconcile this expectation with its input, and sends up only those features of the data that were not expected, a principle now known as predictive coding[27]. So instead of only notifying that there is a mismatch, it reports exactly what features were not accounted for. The higher area can use this information and try to revise its interpretation in the right way, while at the same time trying to comply with the expectations from the even

[26] David Mumford (1992)

[27] Yanping Huang and Rajesh Rao (2011). Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593

higher levels. The templates are to be flexible, incorporating some uncertainties in their prediction in order to account for the variations of patterns, or equivalently in order to allow the higher level to ignore details[28].

[28] More on this in section §5.7.

Theories like these have led to the belief that, rather than a 'cognitive couch potato'[29] that passively processes incoming signals, the brain is a proactive system that attempts to model the causes of its sensations, simulate the external world, and generate expectations for its inputs. The actual inputs are compared to the expectations, and mismatches are used to revise the simulation's assumptions about the current situation, so in non-trivial situations inference proceeds iteratively until a suitable interpretation has settled. When seeing generation as the brain's primary activity, the diagram of the cortical hierarchy could equally well be turned upside down so that the feed-forward connections are regarded as the feedback connections, providing a corrective signal that makes the generation stay in sync with the real world.

[29] Term stolen from Andy Clark (2014)

Andy Clark (2014). Bayesian predictive coding. Recorded talk in a Greenland trip organised by the Moscow Center for Consciousness Studies

The belief that the brain processes its input by continuously making predictions has found wide support, and it is considered quite likely that some kind of predictive coding mechanism is an intrinsic part of this predictive processing. Outside the academic world, books like On Intelligence by Jeff Hawkins[30] have popularised the main ideas, as have many optical illusions and tricks that reveal how much human perception is primed by context and presupposition.

[30] Jeff Hawkins and Sandra Blakeslee (2007). *On intelligence*. Macmillan

3.4.2 Supporting neural phenomena

Besides arguments based on cortical connectivity and behavioural psychology, activity measurements from neurons in the cortex also provide hints about the role of descending connections in prediction. One of the neurological observations that suggested an important role for feedback in perception is the extra-classical receptive field, which alludes to cases where the response of a neuron or column is influenced by signals that it was not considered sensitive to, i.e. signals outside its receptive field. For example, a line detector in V1, that activates when a line segment with a particular orientation is present in a particular location, appears to reduce in activity when the line continues beyond its view. This quirk, called the end-stopping effect, could be explained by postulating that the activity reports the unexpected appearance of a line segment, and is therefore silenced when it is part of a longer line.

In the late 1990s, Rajesh Rao and Dana Ballard demonstrated that certain extra-classical receptive field effects, most notably the end-stopping effect, can be reproduced by a predictive coding algorithm[31].

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

In their architecture, units in a lower hierarchical level see only a small area of the input, giving each higher layer unit a larger receptive field. The error signal a level sends up consists of each unit's activation minus the prediction it receives from above. The activations at the bottom are just the pixel values, and at each following level the activations are determined by minimising both the incoming and the outgoing error signals. A prediction is formed by a linear transformation of the level's activations, possibly followed by a sigmoidal non-linearity. The weight matrices are trained to minimise the average residual prediction errors when given natural image patches as input. The end-stopping effect can be demonstrated by passing in (unnatural) images of a bar of different lengths. Because in natural image patches lines tend to continue, the learnt basis features are less capable of generating short lines, making that short bars leave a higher residual error. For the unit shown in the plot, the responses qualitatively correspond to activations of layer 2/3 neurons in the primary visual cortex.

Although the described model is unrealistically simple, its ability to exhibit phenomena observed in the cortex increases the plausibility that the cortex applies some form of predictive coding. Besides the extra-classical receptive field effects, several other measured phenomena could be explained by predictive processing. In particular, transient neural activity upon changing input seems to match well with the process of updating one's interpretation, after which activity is quenched by resulting new predictions. From brain activity measurements (e.g. EEG), it is known that an 'oddball' stimulus can elicit significant electrical responses, such as the Mismatch-Negativity (MMN) or P300 response, commonly triggered by inserting infrequent deviant tones in a repetitive tone sequence. The brain appears able to learn to expect deviations, matching closely with what hierarchical predictive coding theories would suggest to happen[32].

[32] Catherine Wacongne, Etienne Labyt, Virginie van Wassenhove, Tristan Bekinschtein, Lionel Naccache, and Stanislas Dehaene (2011). Evidence for a hierarchy of predictions and prediction errors in human cortex. *Proceedings of the National Academy of Sciences*, 108(51):20754–20759

3.4.3 *Probabilistic views*

Transcending the implementations and mechanisms, many researchers have proposed theoretical frameworks to understand how (human) intelligence works. A significant group uses probability theory to interpret the brain as a mechanism that perceives by performing (or approximating) Bayesian inference in a probabilistic hierarchical generative model, and that learns this model by applying empirical Bayes. In the Bayesian inference perspective, the descending connections in the cortical hierarchy are interpreted as communicating prior probabilities for the values of the targeted lower layer, which combines this prior with information from the ascending pathway to infer the likeliness of its possible values.

This view is still too vague to pin down an exact meaning of neural signals, whose spiking activity could perhaps encode the probabilities of values, the most likely value, or other distribution parameters. If it is presumed that the cortex applies predictive coding, the ascending pathway should in some way convey the unexpectedness of the inferred values with regard to their prior, so that the neurons will be silent when the higher levels have formed a correct interpretation.

Not unlike a Monte Carlo Markov Chain (MCMC) method, cortical regions concurrently update their values, send new priors down and error signals up, and thereby form a recurrent process that should lead to the system as a whole converging to a consistent interpretation, i.e. a set of values with high probability given the sensory input[33].

[33] Tai Sing Lee and David Mumford (2003). Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448

Learning and inference can be elegantly combined in this probabilistic interpretation. The inference process can be viewed as performing the expectation step of an expectation–maximisation algorithm. The maximisation step is the way of learning the synaptic weights, which, after having inferred each level's values, updates the ways as to make the current state more probable[34]. The optimisation objective can be formulated as reducing the so-called 'free energy', a concept borrowed from statistical physics, which here amounts to the total prediction error[35]. Inference thus tries to find the best interpretation that minimises the prediction errors for the given input, while the learning process tries to find the best synaptic weights to minimise prediction errors averaged over a longer time scale.

[34] Rajesh Rao and Dana Ballard (1997). Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural Computation*, 9(4):721–763

[35] Karl Friston (2005). A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836

Models like these attempt to explain the functioning and architecture of the brain by regarding them as an implementation that pursues a single, information-theoretic goal. Extensions to these theories incorporate action into the model, explaining reflexes and behaviour as resulting from the very same objective of reducing prediction error, and perhaps try to include some explanation for consciousness[36].

[36] Andy Clark (2013). Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(03):181–204

3.5 *Untapped inspiration*

Although discoveries from brain research have provided quite some inspiration for the creation of artificial neural networks, many concepts that seem to be employed by the cortex have not quite found their way yet into the machine learning world. Perhaps the most remarkable difference between the two fields is the current focus in machine learning research on the feed-forward inference network. When algorithms implement generation, it is usually separated from inference, and when they apply recurrence the ideas of building a hierarchy are often left out.

In chapter 5, we will use the just described ideas about iterative

Bayesian inference in the cortex in order to come up with improvements for the way we approximate inference in probabilistic generative models. But first, we leave the networks themselves mostly unchanged, and devote a chapter to the role that downward connections could have in learning the connection weights.

4

Generation as feedback for learning

Practically every deep learning method uses a feedback path, defined either explicitly or implicitly, to solve the central problem of credit assignment: how to tweak the network's parameters (weights) in order to improve the model. Most algorithms perform inference in a single feed-forward pass, but to determine useful parameter updates in every layer they back-propagate gradients of their cost function down the network.

In this chapter, we take a better look at back-propagation and its issues, and consider alternative methods of providing feedback signals for learning. We will find that adding an extra path for back-propagation feels overcomplicated and unnecessary for unsupervised algorithms that already incorporate both a bottom-up inference and a top-down generation path, so in particular we focus on methods that use those existing paths to provide the learning signals for each other, leading us to the existing ideas of target propagation and recirculation, and to more general insights about alternative and local approaches to learning.

4.1 Back-propagation of gradients

In most neural network design nowadays, parameter update rules are specified only implicitly, by defining an objective function and assuming that a gradient-based optimisation algorithm will determine good updates. In contrast to earlier days, in which learning rules would be designed more directly, attention now seems to go to the design of the inference network, and learning almost seems to come as an afterthought. To obtain a connectionistic model incorporating both learning and inference, we can work out the optimiser's update calculations to reveal the complete network of computations involved in learning, which can be more fitly compared to neural networks in the brain.

4.1.1 *The gradient calculation*

For the purpose of obtaining a simple network diagram, we assume that input data are processed one by one, thus performing online learning or stochastic gradient descent (SGD) with a mini-batch size of 1. In practice, hardly anybody uses plain SGD, and an abundance of variations have been developed, ranging from adding momentum to invoking batch normalisation, but the basic principle of gradient calculation usually remains the same[1]. To start with, let us take the following three-layer MLP:

[1] Note that for example second-order methods are not considered here, and are quite a different case; however they are less popular due to their computational cost.

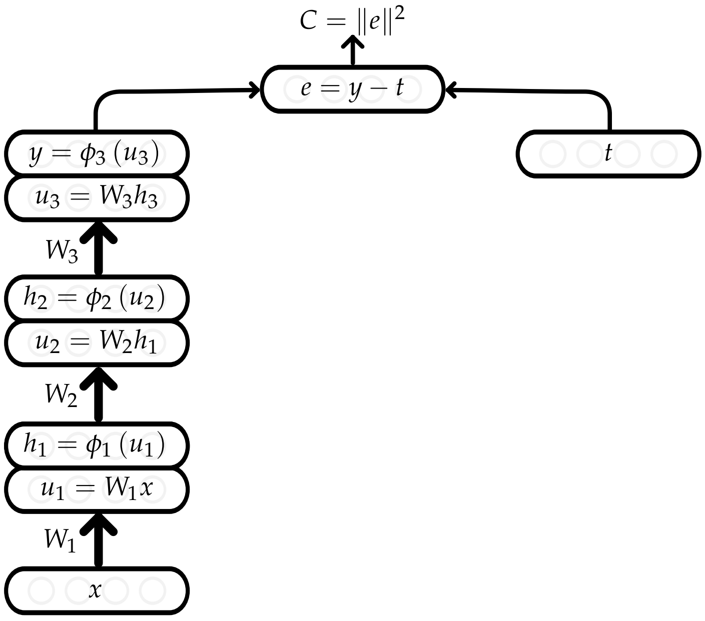

Figure 4.1: A simple MLP with the squared error between the output value y and a given target value t as its cost function. The activation before the non-linearities ϕ_l are denoted as u_l.

The updates for the weight matrices can straightforwardly be derived using the chain rule for derivatives. The weight update for W_3 is computed by taking the gradient of the cost function to W_3:

$$\Delta W_3 \propto -\frac{\partial C}{\partial W_3} = -\frac{\partial u_3}{\partial W_3}\frac{\partial C}{\partial u_3} = -\frac{\partial C}{\partial u_3}h_2^T = -\frac{\partial y}{\partial u_3}\frac{\partial C}{\partial y}h_2^T = -\phi_3'(u_3)\, e h_2^T$$

And likewise updates are computed for W_2 and W_1, which each reuse part of the gradient computation in of the layer above it, hence the name back-propagation:

$$\begin{aligned}\Delta W_2 &\propto -\frac{\partial C}{\partial W_2} = -\frac{\partial u_2}{\partial W_2}\frac{\partial C}{\partial u_2} \\ &= -\frac{\partial u_2}{\partial W_2}\left(\frac{\partial h_2}{\partial u_2}\frac{\partial u_3}{\partial h_2}\frac{\partial C}{\partial u_3}\right) \\ &= -\left(\phi_2'(u_2)\, W_3^T \frac{\partial C}{\partial u_3}\right) h_1^T\end{aligned}$$

$$\begin{aligned}
\Delta W_1 &\propto -\frac{\partial C}{\partial W_1} \\
&= -\frac{\partial u_1}{\partial W_1}\frac{\partial h_1}{\partial u_1}\frac{\partial u_2}{\partial h_1}\frac{\partial C}{\partial u_2} \\
&= -\phi_1'(u_1)\, W_2^T \frac{\partial C}{\partial u_2} x^T
\end{aligned}$$

Drawn into the diagram, this computation proceeds from the top to bottom layer, forming a feedback path opposing the inference path and telling the lower layers how to change in order to satisfy the objective at the top:

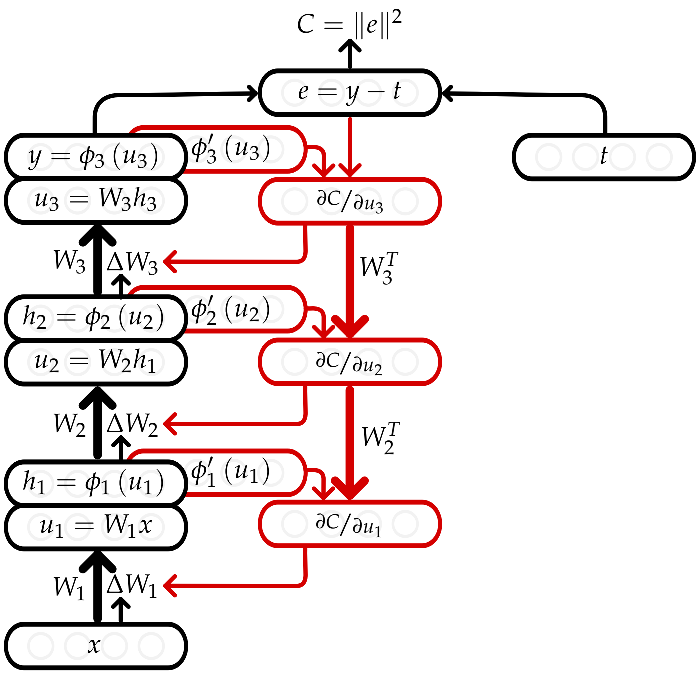

Figure 4.2: The same network as in the previous figure, but now including the back-propagation path in the diagram.

4.1.2 *Issues with back-propagation*

Although back-propagation has proved very useful for many tasks, it has a few known, interconnected problems. When networks become deeper, gradients have to pass through many layers and tend to decay or explode, making it hard to perform learning properly in the lowest layers. Since back-propagation measures only a very local direction of improvement, only small update steps can be taken, and large numbers of iterations are required for learning, taking lots of computational power, as was mentioned in the introduction).

To keep gradients at reasonable magnitudes in a deep network, layers are required to behave sufficiently linearly, as e.g. a saturated response of neurons kills the gradient. The use of back-propagation thus imposes limitations on the design of the inference network, hindering the use of saturation, neural competition, recurrence and other strongly non-linear behaviour. Moreover, it has been shown that networks with a high degree of linearity inherently suffer from undesired extrapolation effects, making that data with unnoticeable amounts of carefully crafted noise (so-called 'adversarial' examples) can confuse the inference process[2,3].

[2] Ian Goodfellow (2015). Do statistical models 'understand' the world? Talk at the RE.WORK deep learning summit 2015

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

Insight: Relying on back-propagation for learning limits the freedom of network design.

Besides practical issues, a frequently reported annoyance of back-propagation is its biological implausibility:

"Many neuroscientists treat backpropagation with deep suspicion because it is not at all obvious how to implement it in cortex."[4]

[4] Geoffrey Hinton (2007b). To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547

One reason for the suspicion is that because in order to compute the gradients, the feedback path needs to know the synaptic weights of the inference path, and it is deemed impossible that the brain could be using this method, as in some way a 'weight transport' would be required to keep the two paths matched[5]. This observation has led many to seek approximations to back-propagation that could provide more plausible explanations for learning in the brain, for example by loosening the symmetry requirements[6], applying contrastive Hebbian learning[7], or using completely random fixed weight matrices in the feedback path[8].

[5] Stephen Grossberg (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63

[6] Qianli Liao, Joel Leibo, and Tomaso Poggio (2015). How important is weight symmetry in backpropagation? *arXiv:1510.05067*

[7] Xiaohui Xie and Sebastian Seung (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural computation*, 15(2):441–454

[8] Timothy Lillicrap, Daniel Cownden, Douglas Tweed, and Colin Akerman (2014). Random feedback weights support learning in deep neural networks. *arXiv:1411.0247*

Remarkably, quite some of these alternative methods seek to modify MLP learning to provide a plausible learning mechanism for the brain, but while doing that still assume the presence of a desired output value for each input, implying that the brain would implement supervised learning. Since brains normally do not learn specified input-output mappings, it may be more fruitful to instead investigate methods for unsupervised learning. Of course, one could argue that supervised methods can be applied for unsupervised learning by setting the output target value equal to the input value, which is the essence of an auto-encoder. However, any method similar to back-propagation seems a contrived and unnatural way of training auto-encoders, because it creates two mostly separated networks, and introduces a second bottom-up and a second top-down path to provide the feedback for learning (see figure).

Learning signals thus have to travel back all the way through the decoder and encoder, because only at the very output layer an objective is specified (plus possibly a regularisation objective in the middle). While in supervised learning it may make more sense that the whole network is adjusted in order to please the output value, in auto-encoders producing the target value is not the primary

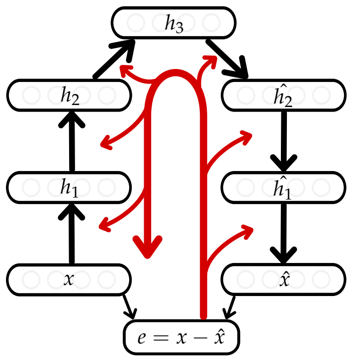

Figure 4.3: In an auto-encoder, back-propagating errors all the way up and down again looks like a needless detour.

goal, which perhaps permits different ways of approaching the problem. Instead of regarding the encoder and decoder as completely separate networks, the symmetry of encoder and decoder should be exploited to treat corresponding encoder and decoder layers as different 'sides' of the same layer. More generally stated, it even feels somewhat narrow-minded to apply supervised learning methods to unsupervised learning.

Insight: Sticking too close to supervised learning techniques when doing unsupervised learning hampers free thought.

Further below we will consider spreading out the objective through the network, and then getting rid of the back-propagation path altogether, but first we ask ourselves whether and why we actually want feedback for learning.

4.2 *Why need a feedback path?*

It has been tacitly assumed that in deep learning, some kind of feedback would be required to teach the lower layers to please the higher layers. In supervised learning, this principle makes sense because the higher layers have very specific targets, whereas the lower layers have no clue what is desired of them. In unsupervised learning, with the purpose of disentangling concepts to form more useful data representations, the requirement is not so obvious. An interesting and very fundamental question is whether or why such feedback would be needed at all.

4.2.1 *Learning without feedback*

To commence in support of the opposite premise, it is indeed possible to do unsupervised deep learning without top-down feedback. This is what happens in greedy algorithms, in which layers optimise for a local objective without taking subsequent layers into account, so higher layers are unable to influence the learning in the layers further below. These algorithms commonly proceed by starting with a shallow, one-layer network, and training it on the data to learn a new representation. Then these representations are treated as the new data for a next layer that is put on top of the first, which learns to disentangle this representation a bit further, while leaving the bottom layer fixed. This process can be repeated to build a deep network as a stack of individually trained one-layer networks.

Greedy layer-wise training sparked interest in deep learning in 2006, after unprecedented learning results were obtained by stacking restricted Boltzmann machines (RBMs) to pre-train deep belief nets (DBNs)[9]. This inspired related work such as stacked auto-encoders[10] and stacked denoising auto-encoders[11], but quite soon layer-wise (pre-)training lost popularity due to improved results with plain back-propagation[12].

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

[10] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153

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

[12] e.g. Dan Claudiu Ciresan et al. 2010; Alex Krizhevsky et al. 2012

Another notable example of layer-wise training in unsupervised learning is the application of three layers of topographic ICA on images, in which — without subsequent supervised training — some units in the highest layer appeared to respond to the presence of a human face or of a cat face in an image[13]. Like normal ICA, a TICA layer extracts least-dependent (ideally independent) components, but also pools the components with strong residual dependencies (higher-order cumulants) together. The experiment nicely demonstrates that plainly removing the statistical dependency structure from the data, step by step, can reveal variables that correspond to abstract concepts. A usable metaphor for the approach is that the structure in the data is peeled off like the layers of an onion, the removal of one layer revealing the next.

[13] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng (2011). Building high-level features using large scale unsupervised learning. *arXiv:1112.6209*

Insight: An elegant objective for unsupervised learning is to peel off the predictable structure from the data.

4.2.2 *Lower layers lack knowledge*

Despite occasional utility, most often for pre-training, greedy layer-wise training is ultimately not very promising. The fundamental issue is, as it is more often with greedy algorithms, that doing the best possible step at every layer does not necessarily lead to the best end result. For the higher layers to produce an optimal result, the lower layers may need to prepare the data in a way that, at their layer, is not clear why it would be a good choice. Therefore, knowledge about how the subsequent steps are doing may be required to reach the higher goal.

The way this issue manifests in unsupervised learning, is that a relation between patterns may not be detectable in a lower layer, while that lower layer does need to extract those patterns from the data. Only when the structure has been detected it may be known how to expose it better, so the lower layer needs to hear from above which patterns to extract. Limitedness of resources plays a role here, since lower layers may have to choose which patterns to spend their units on, while only at higher layers it will be known which ones are relevant. Seen in another way, a lower layer may have many possible ways to peel off the structure, that may for that layer's objective seem equally satisfying, but leave the data in a different state for the next layer. Without top-down information, the result could depend even on tiny differences in its initial conditions, so some guidance from the layer above would direct it to choose the option that best reveals the deeper structure.

Insight: Lower layers are unaware of higher-order relations, so they need guidance to choose in which way to peel.

Altogether, the case for unsupervised learning may actually not be that different from supervised learning. In principle, we should be able to use an unsupervised learning technique to do supervised learning by giving the label as an extra input, possibly directly to a high layer. For example, we could give a single bit indicating 'happy' or 'sad' along with images of faces. For the system to figure out as much as possible of the structure in the data, it should learn to extract emotional indicators in its lower layers, since they correlate strongly with the supplied mood bit.

A nice observation is that also without manually providing it as an input, mood is likely to emerge in a latent variable because it has predictive power over the whole face, and top-down feedback will then be useful to refine indicative patterns, to both increase and exploit the predictive power.

4.3 *Higher layer objectives*

The issues with back-propagation learning arise from the fact that layers try to satisfy an objective on which they have only very indirect influence. The obvious way to ameliorate this problem is to originate learning signals closer to the layers on which they act, by providing objectives throughout the network rather than at a single spot. In the auto-encoder architecture, this can be achieved by exploiting the symmetry between the encoder and decoder paths. Rather than only at the bottom, one can for instance demand that at every layer h_l the difference between encoder and decoder values is minimised:

$$C_l = \left\| h_l - \hat{h}_l \right\|^2$$

Effectively, the subnetwork above (and including) that layer is then treated as an auto-encoder, and the decoder learns to reconstruct the encoder's value at every layer. Besides, since unlike at the bottom layer the encoder's values are computed rather than fixed, the encoder also learns to produce representations that the decoder can reconstruct well, although a regularisation objective may have to be added to prevent it from learning trivial (e.g. constant) representations.

4.3.1 *Composed into a single objective*

Higher layer objectives can be used in combination with back-propagation, by combining them into a single cost function and using this as the objective for all parameters. The cost function is then the sum of the errors at each level, including the original overall reconstruction error at the bottom. The contributions of the layers can be scaled by constant factors to set their relative importance, normally to give more weight to the errors at lower layers:

$$C_{total} = \sum_{l<L} a_l C_l$$

In the computation graph, each layer injects its error into the feedback path, so that by the linearity of derivatives the gradient at every layer is the desired weighted sum of the individual objectives' gradients with respect to that layer (see figure 4.4). The result of the approach is that if the gradient carrying the overall reconstruction error has decayed strongly, the layers still receive a significant learning signal from the layers close by, guiding them to make the subnetworks good auto-encoders, and thereby speeding up the learning process.

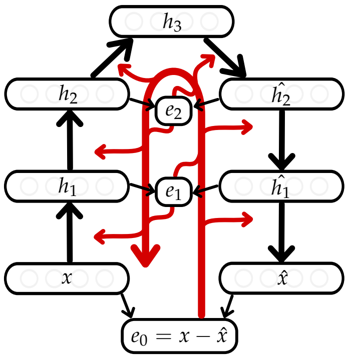

Figure 4.4: Each layer's reconstruction error contributes to the back-propagated gradient (compare with figure 4.3).

An example application of higher layer objectives is the Ladder network[14], which forms a deep denoising auto-encoder in which each layer has the objective to denoise a noisy version of the encoder's value.

[14] Harri Valpola (2014). From neural PCA to deep unsupervised learning. *arXiv:1411.7783*

4.3.2 *Using only local objectives*

An interesting alternative to the approach of optimising every layer with the same composition of all layers' objectives, is to use only the local objective for the training of each layer. By simply not applying the chain rule of derivatives, this approach yields local learning rules. This means that each layer updates its weights using information available in that layer, and thus completely removes the need for the back-propagation path.

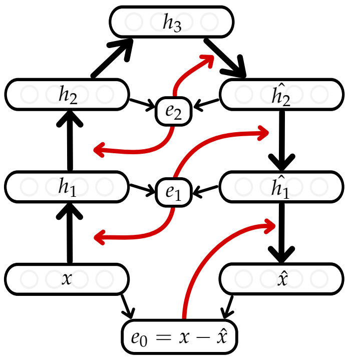

Figure 4.5: Removing the back-propagation path in figure 4.4 lets each layer learn with only its local objective (although the top layer h_3 lacks an objective in this particular case).

The big question in this case is whether these independently learning layers will together still form a coherent model. As argued earlier, for a high layer to perform optimally some feedback to the lower layers would be desired in order to teach them how to serve its needs. Back-propagation of an objective function gradient would be the obvious solution to communicate its needs, but perhaps local learnings rules can also be made 'globally aware' by including information from above in their objectives, and this the approach we will investigate. The just proposed higher layer objectives, although simplistic and unrefined, do incorporate information from above by depending on the reconstructed values.

To ensure a formal justification for a local update rule, ideally a layer's gradient of its local objective would always point in the same direction as the gradient of the global (composed) objective, in which case using the local learning rules would be equivalent to using back-propagation with the composed objective. A similar but easier goal would be to require that the update of each parameter at least has the same sign as it would have had when using back-propagation[15]. Even more lenient, but likely still sufficient, would be to just keep the angle between the two directions below 90°, so that updates still ascent towards the objective, although perhaps not taking the *steepest* ascent[16]. Remember as well that steepest descent is a greedy form of optimisation, and updates deviating from the 'ideal' direction could in fact be superior.

[15] Yann Le Cun (1986). Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer

[16] Timothy Lillicrap et al. (2014)

4.4 *Generation as feedback*

The main question in this chapter is whether the generation path can simultaneously serve as feedback for the inference path, providing the information required for the lower layers to determine how to better please the higher layers. Admittedly, this quest for using a single path for two purposes is partly driven by a desire for elegance and biological plausibility, as it is not evident that a model containing multiple paths necessarily poses a problem. The currently ubiquitous approach of gradient back-propagation does however pose problems, as has been argued above, so integrating the generative model more heavily into the learning process seems a logical thing to try. There have been several algorithms and ideas with similar intent, a few of which are worth noting here.

4.4.1 *Wake-sleep algorithm*

A good method to look at first is the wake-sleep algorithm, a simple and intuitive algorithm to learn two directed probabilistic models, one for generation and one for inference, by alternately teaching either to invert the other. In the wake phase, a datum is passed upwards through the inference model, and the generative model is updated to, at each layer, make it more likely to generate the lower from the upper value. In the sleep phase, a 'fantasy' value is sampled from the generative model, and the inference model is updated to more probably infer the fantasy's causes. Because each layer learns individually to invert one step of the opposite path, the updates are completely local, although the algorithm is not greedy.

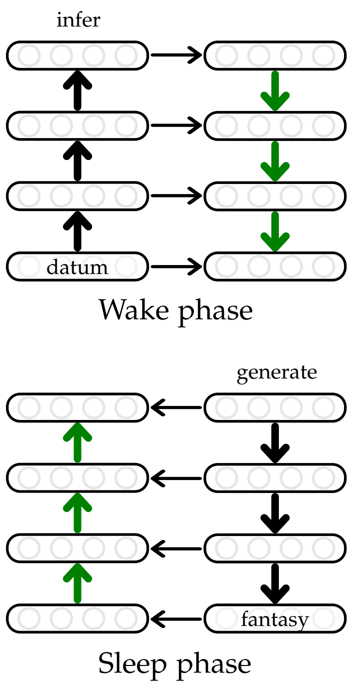

Figure 4.6: The two phases of the wake-sleep algorithm. The green arrows are updated to, at each layer, learn to invert the result produced by the other model

A variation of wake-sleep, called up-down, has been used for fine-tuning deep belief nets[17]. Instead of starting the generation pass with a completely random sample at the top layer, it samples it with a bias towards the top level value in the inference path[18]. If, instead, that top level value would simply be copied from the inference top layer, the resemblance with variational auto-encoders becomes apparent, while the training method can be interpreted as a form of target propagation.

[17] Geoffrey Hinton et al. (2006)

[18] More specifically, sampling proceeds by a few iterations of Gibbs sampling in the RBM formed by the two topmost layers.

4.4.2 *Target propagation*

The idea of target propagation is to provide, instead of a gradient towards an objective, a target value to each layer that would help achieve an objective at the top. Intuitively, if the inverse function of an inference network is available, it is possible to compute which values of lower layers would have resulted in a desired value at the top. In other words, the target value is propagated downwards using the inverse functions, and each layer locally computes a weight update to steer its value closer towards its given target. Different from back-propagation, non-linearity and even discrete-valued activation functions do not pose a problem, as long as the inverse function is available for the occurring values. Under simple conditions, it can be shown that this method can yield updates that have the same signs as the updates back-propagation would compute using all layer objectives combined[19].

[19] Yann Le Cun (1986)

The inverse function required for target propagation is exactly what the generation side (decoder) of an auto-encoder approximates. Although in unsupervised learning no explicit target value is given for the top layer, the principles can be applied nevertheless and the auto-encoder can be trained by providing target values for each layer, an idea hatched by Yoshua Bengio[20]. The challenge is to decide on good targets for the layers, which by the way need not be the same for the encoder as for the decoder.

[20] Yoshua Bengio (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv:1407.7906*

To exploit the principle of target propagation, a layer's target for its encoder step should be the value produced at that layer by the decoder (see figure). Because the encoder is (presumably) the inverse of the decoder, when a lower layer adapts towards its target value, it will cause the higher layers to do so too, thereby achieving exactly what we desire of a feedback path.

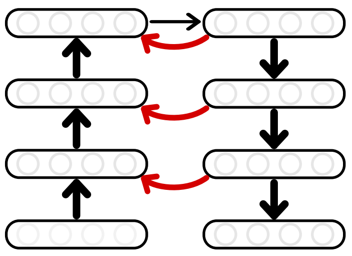
Figure 4.7: The values generated in the decoder are used as targets for the layers in the encoder.

It would make some sense to likewise use the encoded value as the target for the decoder, to ensure that the decoder properly reconstructs encoder values, and thus indeed keeps approximating the inverse of the encoder. In this symmetric case we arrive at exactly the local objectives $C_l = \left\| h_l - \hat{h}_l \right\|^2$ introduced above. Despite its elegance, this choice is probably too naive, and would result in many unhelpful updates when the layers are not yet forming good auto-encoders, so better targets are desired to prevent the learning process from getting stuck.

To be useful as targets for the encoder, preferably the values produced by the decoder would in some way be justifiably better than the currently inferred values. Using the training method from the denoising auto-encoder, noise can be added to the input of the decoder's layers while the target for the decoder is the clean value in the encoder. The reconstructions are then expectedly cleaner (more probable) versions of the values in the encoder, and could be suitable as targets to make the encoder learn to separate information from noise and disentangle the information to make it easier to model in the subsequent layers.

There are still many details to be figured out to implement these ideas. In the learning algorithm sketched by Yoshua Bengio, values are propagated through the decoder cleanly to serve as targets for the encoder, while the decoder is trained to denoise by at each layer also corrupting its input with noise and using the encoder value as target[21].

[21] Yoshua Bengio (2014), algorithm 2, second loop

Moreover, the algorithm alternates learning with these propagated targets with layer-wise training to ensure the layers individually are good and contractive auto-encoders, as well as to provide a way to train the top layer[22]. To do this again without back-propagation, targets for this step are created by a denoising variation of the recirculation procedure, a method described below.

[22] Yoshua Bengio (2014), algorithm 2, first loop

In practical experiments by Yoshua Bengio's research group, it was found that the imperfectness of auto-encoders obstructs learning, but can be accounted for by incorporating a linear correction in what has been named difference target propagation[23].

[23] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio (2015). Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer

4.4.3 Recirculation

In the algorithms we have treated up to now, the units in the generative path could be regarded as related to, but distinct from, the units of the encoder path. If implemented in the cortex, they would be close to each other, probably situated in different cortical layers in the same column. Another option is that the descending path would reuse the *same* neurons that were used in the ascending path. The downward, generative pass would then update the activations that were determined in the forward pass. The idea of the recirculation algorithm[24] is that subsequently a second forward pass could then be used to inform the higher one of two layers about the effect it had on

[24] Geoffrey Hinton and James McClelland (1988). Learning representations by recirculation. In *Neural information processing systems*, pages 358–366. New York: American Institute of Physics

the lower one, and thereby provide it with a learning signal.

In learning a basic, single hidden layer auto-encoder, the decoder weights can easily be updated using gradient descent, since the datum serves as the target value for the reconstruction. The recirculation algorithm suggests to apply the same principle to learn the encoder, by treating the initial encoded value as the target value for the second forward pass (see figure).

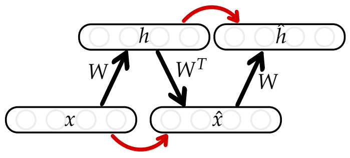

Figure 4.8: In recirculation, the reconstructed value is passed through the encoder a second time.

An additional idea applied in recirculation is to give the bottom units a high level of regression, meaning that their value is changed towards, rather than replaced by, the reconstruction coming from above. This regression makes that the value passed upwards in the second forward pass is not much different from that in the first, so the hidden layer measures the effect of a small change in its input. Because for small changes a network behaves roughly linearly, the method effectively measures the gradient and can approximate gradient descent on the reconstruction error. One remaining requirement for this to work is that the encoder and decoder weights are symmetric. Luckily, the learning procedure is self-aligning, so unmatched weights will automatically adapt to produce symmetry.

4.5 *Learning through recurrence*

The old and largely forgotten recirculation method, and its modest revival in target propagation, can provide inspiration for creating globally-aware local learning rules, like was desired in section §4.3.2. Although the method is described for a simple auto-encoder with one hidden layer, we can reuse its core principle, which is to learn locally from the differences between old and new values in what essentially is a recurrent neural network formed by the inference model and generative model together. This principle has for example led to the 'generalized recirculation algorithm'[25], and will form the basis of the learning paradigm assumed throughout the rest of this thesis.

[25] Randall O'Reilly (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938

4.5.1 *Local learning signals from temporal difference*

Since the use of a (vector) target value for a layer in fact applies a scalar target value to each unit individually, the principle of learning from temporal value changes is easily seen to be a generalisation of the Hebbian learning rule. Hebbian learning strengthens connections between activated neurons and their active inputs ("neurons that fire together, wire together"), or, in other words, a neuron's current output value is used as the target for its current input. The rule can be generalised, such that current as well as previous activations can be targets for either current or previous inputs.

We can compare how learning algorithms differ when applying the assumption that their inference and generation paths reuse the same units. In recirculation, in both layers their previous output value is the target output for the current input, thereby measuring the imperfectness of the auto-encoder and learning to reduce it. In target propagation as described above, this same rule still basically holds for learning the decoder, but the encoder appears to learn the other way around: the new (top-down generated) value of a layer is used as the target for the previous input from below. Intuitively, the encoded value in a layer can be considered as the initial impression, and the reconstructed value as the refined interpretation after having fully processed the input. By using the refined interpretation as the target for the encoder, it should learn to infer the refined interpretation directly when given similar input in the future.

Insight: Future value as target. The core learning objective of a unit in a recurrent network could be to directly obtain the value it will settle to later.

4.5.2 Neural plausibility

In the brain, learning from temporal differences is a very plausible learning mechanism, as has for example been suggested by Geoffrey Hinton[26]. An observed phenomenon in neural synapses is spike-timing-dependent plasticity (STDP), which means that the change in synaptic strength depends on the relative timing of the spikes of the post-synaptic and pre-synaptic neuron. Commonly, the synapse grows in strength most strongly when the post-synaptic neuron fires within milliseconds after the pre-synaptic neuron, which matches with Hebbian learning since the pre-synaptic spike was contributing to triggering the post-synaptic neuron[27].

[26] Geoffrey Hinton (2007a). How to do backpropagation in a brain. Invited talk at the NIPS 2007 Deep Learning Workshop

[27] Wulfram Gerstner et al. (2002), chapter 10.

An interesting variation to Hebbian learning can result when a slightly longer time difference still invokes plasticity. The effect would be that the synapse grows when the pre-synaptic spike did not actually contribute, but it *could have* been listened to in order to fire earlier. Like suggested above, neurons then learn to directly acquire the state they will settle to later, in other words they learn to predict their own activation[28]. Another interpretation of this effect is that neurons learn to keep their output constant upon changing input. This reminds of slow feature analysis, and is an intuitive way of creating invariances. Invariances that are not necessarily temporal may also be learnt this way, for example translation invariance could be learnt because objects (or eyes) move.

[28] Yoshua Bengio (2014), section 8.

Staying close to usual auto-encoders, in this chapter it has often been assumed that we receive single inputs, and perform the generation pass after an inference pass. When dealing with a continuous stream of input, as is the case in brains and for example robotics, it makes sense to consider generation to happen alternately or simultaneously with inference. Although we will not go into it here, the discussed methods could be adapted to use a reconstruction from previous input as a learning signal for the current input.

4.5.3 Amortised inference

One sensible way to create target values for the inference model, would be to perform more accurate inference with an iterative optimisation procedure, and use its result as a target for the approximate inference model. This approach follows the idea of amortised inference algorithms, which could be described as "learning how to do inference faster next time". Predictive sparse decomposition (section §2.4.2) fits in this category, and more sophisticated approaches have tried to make the approximation more powerful, by designing the inference model's architecture such that it resembles the computation done by optimisation procedures[29]. Such a network would be recurrent and trained by back-propagating through time, where to improve learning the target value could be assigned to every time step, to encourage the network to arrive at the optimal interpretation as fast as possible[30].

[29] Karol Gregor and Yann LeCun (2010). Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 399–406

[30] Ulugbek S Kamilov and Hassan Mansour (2015). Learning optimal nonlinearities for iterative thresholding algorithms. *arXiv:1512.04754*

> **Insight:** A slow optimisation inference procedure can provide the target value for learning a fast approximation.

In the following chapter we will look how the generative model itself can be leveraged to perform more accurate inference, and in the synthesis we will shortly get back to how to use the hereby inferred values as local learning targets for the layers.

4.6 The curse of deep learning

Before finishing this chapter, one more point to bring up is that some of the intuitions behind the deep learning methods covered here may be valid only when the higher layers of the model are assumed to already have learnt some adequate parameters. In case of the wake-sleep algorithm, the generative model initially produces complete nonsense, which the inference network then attempts to invert, so it can take a long time before the system learns something reasonable. At the bottom layer the generation will learn to produce the actual data, but it will produce them from the random fantasies of the layer above, whose distribution will change when the network above learns. There is something unsatisfying in the method of learning because it seems fundamentally inefficient, if it works at all without getting stuck.

Target propagation suffers similarly from the problem, as it relies on the higher layers to produce good targets for the layers below them. As the higher layers completely determine lower-level values, whatever the lower layers learn initially will be based on a state of the higher layers that will change itself. This struggle between learning higher layers while not forgetting what was learnt in the lower layers may be a reason for the large amount of iterations usually required for learning. The issue also occurs when training a deep auto-encoder with back-propagation, because there the initial updates to a low layer in the encoder are also calculated to benefit the objective under the assumption that the layers above remain unchanged.

Given these inherent inefficiencies, the idea of greedy layer-wise training starts feeling quite attractive again, and it is not surprising that greedy methods have proven useful for pre-training. My personal feeling is that a learning algorithm should, without being greedy, be designed such that adding extra layers does not complicate or slow down the learning of the lower layers. Lower layers should be able to function without the higher layers, but adding higher layers should improve their performance. A higher layer should perhaps not assert any influence initially, and gradually start mixing in when it has learnt some patterns and has something to say. In probabilistic models this could for example be achieved by letting higher layers initially only set weak priors on their layers below[31], while in deterministic deep auto-encoders lateral connections between encoder and decoder should initially provide the primary influence. Although more thought may be required to concoct a concrete solution, it may be good to keep the basic concept in mind.

[31] Related to modelling variance, in section §5.7.1

Insight: Low levels need to learn first. Higher layers should initially have no influence, and start participating when they have become more confident after learning a bit.

5

Generation as feedback for inference

In most popular neural network algorithms, inference is performed in a single feed-forward pass. The proactive role of the neocortex in processing its sensory inputs (see section §3.4.1) provides inspiration for how the inclusion of feedback signals could benefit inference. Whereas several methods mentioned in the previous chapter did suggest possible roles for the descending connections in the cortex, they all used these connections *only to provide learning signals* to lower layers of the inference model, and neglected the whole idea that top-down signals could serve a role in input processing itself.

In this chapter we will investigate how feedback can be useful when approximating inference in a learnt directed generative model. We will find that using the downward connections of the generative model in close concert with usual upward connections of the inference model leads to a substantially different approach to inference, forming structured recurrence and processing inputs iteratively. Although recurrent networks are more commonly applied when the input is a time-varying data stream, for the sake of simplicity, as well as to stay close to previously described models, we will generally assume we are dealing with static input, for example a single image. To start with, we will look at the reasons to depart from feed-forward inference.

5.1 Why not feed-forward?

It is worth spending a moment to ponder about the fundamental question why feed-forward networks would be suboptimal for inference, which after all is intrinsically about forming a representation in the higher layers of a datum fed in at the bottom. A first, somewhat general answer is that adding a feedback loop effectively creates an infinitely deep network, and is thereby in theory capable of producing more complex behaviour[1]. Ideally the inference network would perform accurate Bayesian inference in the generative model

[1] In theory, one could build a turing machine (although with finite tape) with a recurrent net (Heikki Hyötyniemi, 1996).

Heikki Hyötyniemi (1996). Turing machines are recurrent neural networks. *Proceedings of STeP*, pages 13–24 and thus compute $p(h|x)$, which is commonly much more complex than the generation process $p(x|h)$ that the model defines explicitly. By making the inference network more powerful, it can better approximate ideal inference, thereby in turn enabling us to learn better generative models[2]. Adding complexity to increase modelling power could of course also be done by simply adding more units and layers to a feed-forward net, but learning to use this power is hard. Since we have knowledge about the problem we face, a more intelligent approach is to exploit that knowledge in the network design.

[2] See insight "q restrains p", §2.2.2.

Insight: We do not want to approximate an arbitrary input-output mapping; we want to approximate Bayesian inference. Compared to an MLP, a neural network structure designed with this task in mind may prove both more successful and easier to learn.

One piece of knowledge we could exploit is the structure of Bayes' rule, $p(h|x) \propto p(x|h)\, p(h)$, which allows us to reason about the individual influences of the separate factors $p(x|h)$ and $p(h)$. Moreover, perhaps the most valuable asset that we can exploit is the availability of the generative model itself. Although the generative model cannot tell us which h probably caused a given x, when we *propose* an interpretation h it can validate whether it matches with this datum, and maybe we can even extract information about how to improve the proposal. We could try to take advantage of this possibility, let the generative model provide feedback for the inference path, and perform inference by the iterative alteration and verification of proposed interpretations rather than by a single feed-forward pass.

Upgrades to the network architecture could enable it to learn computations that resemble message passing algorithms for probabilistic models. When dealing with a deeper generative network, inference becomes increasingly difficult, and provisioning the network for the task therefore becomes both more valuable and more complicated. The idea of sending proposals up and feedback down could be applied between every pair of subsequent layers, but now, since the values of the intermediate layers are not fixed, feedback can be used both to update the proposal sent upwards and to update the layers' own values.

A common reason for *not* employing feedback loops, thus limiting an inference network to be strictly feed-forward, is that the usual learning method of back-propagation requires sufficient linearity of the network to sustain significant learning signals at lower layers (see section §4.1.2), and recurrence effectively creates an infinitely deep network, making it hard to train[3]. Now if we stop relying on back-propagation and design alternative learning methods instead, the inference network can possibly use stronger non-linearities, and recurrence need not be such a problem any more.

[3] Recurrent neural networks are normally trained by back-propagation through time (BPTT), which boils down to 'unrolling' several time steps of the computation to obtain a feed-forward network with shared weights between the 'temporal layers'.

Insight: End-to-end back-propagation learning obstructs the use of recurrence. Adding a feedback loop to inference will require a change of learning algorithm.

To focus fully on the inference procedures we will ignore the aspect of learning in this chapter, so for now we will assume that ideal weight configurations are magically known. Having roughly sketched how inference could be improved for the task of Bayesian inference, the goal is now to make these ideas more concrete, which we will head towards by first analysing the problem of dealing with ambiguous inputs.

5.2 *Solving ambiguity*

Besides requiring a powerful inference model to deal with the complexity of inverting a generative model, a major difficulty in inference is that although on a high level a datum may have only a single sensible interpretation, at low levels and smaller scales, patterns are often highly ambiguous[4]. An enviable ability of the brain is to effortlessly infer the most sensible causes of an input with ambiguities, finding the interpretation that makes most sense for the input as a whole.

[4] Alan Yuille and Daniel Kersten (2006). Vision as bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308

5.2.1 *A visual example*

To better understand what dealing with ambiguity means in terms of Bayesian inference, it may help to introduce a simple example and analyse it. Consider these hand drawn squiggles:

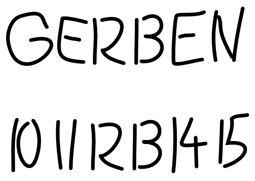

Figure 5.1: Apparently obvious, but objectively ambiguous: The *RB* and *12 13* are exactly the same symbols. Thanks to Harri Valpola for the idea.

Although they look like a string of letters and a sequence of numbers, a better look reveals that the two symbols in the middle of the sequences are exactly the same and could be interpreted either as numbers (*12 13*) or as letters (*RB*)[5]. Humans subconsciously interpret them as to make sense among the surrounding characters, without even noticing the ambiguity. If we assume that some single neuron, let us call it $h_l[R]$, has the task of detecting an R in the center of the input, it appears to not only be affected by whether its input looks like an R. Inferring the cause of the input requires to also take into account whether other explanations are available, and which explanation is

[5] For simplicity two digits are called a single symbol here.

more sensible in this case. The information available on which to base a decision for one of several explanations can conveniently be grouped into two types:

1. The match between the input and each competing feature. Perhaps the symbol could be either a sloppy *12* or an even more crippled *R*, making it being a *12* the more likely explanation.
2. The context outside of the feature itself. Even if the symbol looks a bit more like a *12*, in a word where an *R* would make much more sense, the *R* is the more probable explanation.

There is a clear connection between these types and Bayesian inference. Using the first type of information roughly corresponds to assessing the likelihood of the data given the feature, $p\left(x|h_l[R]\right)$, and should be inferrable from a part of the input below (the unit's receptive field). The second type has to do with approximating the prior probability of the feature, $p\left(h_l[R]\right)$, using the remainder of the input and the generative model above. Combining these two would thus be approximately proportional to the posterior probability, $p\left(h_l[R]\,|x\right) \propto p\left(x|h_l[R]\right) p\left(h_l[R]\right)$, so computing this value for each alternative explanation and choosing the most probable one amounts to doing maximum a posteriori inference.

Of course, this example shows only a simple case where two similar but mutually exclusive features cause the ambiguity. The case becomes more complex when multiple features together form an alternative explanation, or when continuous values rather than binary features are involved. The example may not be overly artificial however, because having binary features that mutually exclude each other seems quite natural when we assume that the generative model uses sparse representations[6].

[6] We will argue for sparsity again in section §5.7.3.

Insight: The use of sparse representations can spawn a lot of units with alternative, conflicting explanations.

5.2.2 Designing for disambiguation

How could we modify an inference network to make it more capable of dealing with ambiguity? From a Bayesian perspective, the idea we will use is to encourage the network to assess the different types of information separately and combine them to form interpretations. Looked at schematically, we could say that Bayesian inference requires four pieces of information to decide between two alternative explanations:

	feature A	feature B
match with input	$p(x\|h_A)$	$p(x\|h_B)$
prior expectation	$p(h_A)$	$p(h_B)$

Inference requires to combine these pieces by doing multiplication within the columns and comparing the values of the columns. To aid the network with doing the comparison, we will first look at how to support lateral competition between alternative explanations. This would allow the features to focus on their own information; to separate, so to say, the columns of the table. Then we move on to the most intriguing question: how to bias these competitions by generating expectations based on the contextual information, thereby effectively enabling units to access the rows of the table separately. We then look how predictive coding uses expectations to filter out expected signals and indirectly provide competition, and combine it with the idea of biasing to get the best of everything.

5.3 *Lateral competition*

As the example demonstrated, the need for competition between possible explanations naturally appears when different causes could have produced the same result, which is fairly common in a world with noise, uncertainty and many non-linear interactions (e.g. think of visual occlusion). Due to this 'explaining away' effect, latent variables in the generative model can become statistically dependent when given the input. In inference, or an adequate approximation to it, the activation of one unit thus implies the deactivation of others, and holding competitions between units that represent alternative explanations is a logical way to obtain this effect[7].

[7] Note that units do not compete with *all* others, which would lead to winner-takes-all or vector quantisation.

Even if strictly taken the input would not be ambiguous, some form of competition is still desirable because units will always be somewhat sensitive to patterns similar to their preferred feature (since features are normally not orthogonal). For example, a *9* and a *g* often look similar, and it would be hard to make each unit's response properties sharp enough to only activate for its intended feature. Moreover, a wide sensitivity may even be a very desirable property, since it helps to generalise to unseen cases.

Insight: Competition is desirable to sharpen or widen units' sensitivity, to improve either discrimination or generalisation, whichever is needed.

5.3.1 Competition in feed-forward networks

First let us look how normal feed-forward networks manage. To start with the pathological case, observe that competition is not possible in single layer inference network, since each unit computes its activation independently of its peers. In probabilistic terms, the units are assumed conditionally independent given the input, implying that the layer has a factorisable probability distribution. It can therefore not adequately infer the posterior of a generative model, unless that generative model is simple enough to have a factorisable posterior[8]. How this handicap influences the features the network will learn will be looked at below.

[8] For example Restricted Boltzmann Machines (RBMs) have a factorisable posterior by design (Geoffrey Hinton, 2000).

Geoffrey Hinton (2000). Modeling high-dimensional data by combining simple experts. In *AAAI/IAAI*, pages 1159–1164

In a feed-forward network with multiple layers, the problem should be less severe because it can learn to simulate competition, since units can subtract activations of units below them to compare their activity and can thereby more or less filter out overridden explanations.

Although probably not as neatly organised as sketched in figure 5.2, presumably contemporary algorithms learn to implement competition to some extent. However, to achieve this the network has to learn to create competition for each set of conflicting units by carefully balancing the biases and positive and negative weights. Since this seems hard to learn, takes many extra units, and the principle is needed ubiquitously, it is tempting to provide the capability of competition between units more directly in the network architecture.

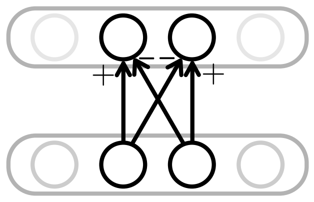

Figure 5.2: A feed-forward network could learn to hold a form of competition by balancing positive and negative weights. Note that for competition between $n > 2$ alternatives, an extra layer and n^2 units would be needed, to compare every pair.

5.3.2 The case without competition

Before mentioning ways of implementing competition in the architecture, it may be helpful to get an intuitive feeling for the need for competition. Imagine a network where units detect possible causes, but that is completely incapable of explaining away alternative causes (e.g. competition is turned off, or units have only positive weights). In this case, the network would probably not be as able to create sharp non-linearities and may behave more similar to a network with a single layer. Multiple alternative explanations would keep propagating upwards, possibly triggering undesired associations with their cross-talk and hindering a univocal and sparse interpretation. Moreover, it might make it harder to perform credit assignment, because erroneously active units should not be encouraged to activate again in that situation, as doing so would prevent units from specialising. This seems most important when we use Hebbian-like local learning rules, in which case a unit does not get to know its influence on the output, but instead its own activation serves as its learning signal[9].

[9] see insight "Future value as target", §4.5.1.

Insight: Lack of competition may hinder credit assignment, especially when learning rules are local.

Not being able to compete severely limits what can be successfully learnt, although the learning algorithm will adapt a bit to the situation. To approximate inference while lacking a way to compete with alternative explanations, each unit will have to attempt to learn features that will let them activate only when its alternatives will not. For optimal detection of a particular pattern, a unit has to learn not just to look for that pattern, but it has to focus especially on the aspects that differentiate its pattern from alternative but similar patterns, in order to not accidentally activate for patterns represented by other units.

Insight: Network architecture influences which features are optimal. Without the ability to compete, units have to learn how their pattern differs from others.

Focussing on what distinguishes a pattern from others may not necessarily appear to be a problem. However, it seems to me that conflating distinguishing features with the pattern itself complicates learning, results in a lack of abstraction and leads to bad generalisation. Looking back at the table presented in 5.2.2, we could say that the network is unable to separate the information from the different columns.

Insight: Units should focus on learning the occurring pattern itself, not on its distinguishing aspects from other patterns, to generalise better and be reusable.

Besides this way of learning to adapt the features, remember that we usually train both the inference model and generative model together, and the generative model can also adapt in order to make its posterior as factorisable as possible[10].

[10] See insight "q restrains p", §2.2.2.

5.3.3 Supporting competition

In a standard feed-forward net, lateral competition between units is technically possible but the network would have to learn how to do it. To help the network with the task, it seems a reasonable idea to let each unit activate when it could possibly explain the input, and quell its activation in case another unit provides a stronger explanation for it. We could say that the units compete for getting the honour of representing the input. Knowing which units should compete can be learnt, or the other way around, units that are hard-wired to compete may automatically learn to represent conflicting explanations.

The competition can be performed by a special hard-wired layer following a normal layer, which can also be seen as an activation function that is not element-wise. A few feed-forward network architectures do incorporate such hard-wired competition. For example, in local winner-takes-all networks, every unit is paired up with a neighbour, and the less active unit of each pair is always muted[11]. Some auto-encoder algorithms apply a form of competition, but then between all units, to enforce sparsity of the representation, as noted in section §2.3.3. Also, the soft-max layer often used in the output of (supervised) classifier networks does hold a (soft) competition to obtain the relative activities of units, but again the competition happens between all units in the layer, and normally only on the final layer.

[11] Rupesh Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber (2013). Compete to compute. In *Advances in Neural Information Processing Systems*, pages 2310–2318

Instead of letting the competition follow each layer as a kind of activation function, another approach would be to apply a simple form of recurrence, by adding inhibitory lateral connections between units, essentially letting each unit's activation threshold be influenced by competing units' activity. Inspiration for such an approach can be drawn from the lateral inhibitory connections in the cortex. By making the lateral connections learnable, it can be configured which units compete, and competition could also be configured to be strong or weak. Competition is very related to representation sparsity, and lateral inhibitory connections may be a good way to enforce sparsity. When choosing prior distributions, it may be worth looking whether an Ising model with negative weights can be used rather than distributions that reduce activities of units individually, like the Laplace distribution.

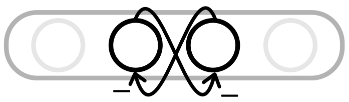

Figure 5.3: Competition through lateral inhibition.

Adding recurrent lateral connections does add the implied complexity of recurrence, so processing becomes iterative since the initial activation of units will compel their neighbours to update their activation, which again changes others' inhibitions, and so forth. A possibly interesting way to approximate or speed up the potentially endless iterations is to first let the most strongly activated units inhibit their competitors before continuing to process the others' activations. This idea is inspired by how in brains, neurons fire earlier when their total received stimulus is stronger, because they integrate inputs until the firing threshold is reached. Neurons firing much quicker might inhibit their neighbours and could prevent them from reaching their firing thresholds at all. Although in a network that is otherwise feed-forward, it may be more attractive to add feed-forward competition, competition through recurrence would not add much extra trouble if we would be adding recurrence anyway; a plan that is discussed next.

5.4 *Biasing competitions with expectations*

Let us get back to the core idea of this chapter, that generation can be used as feedback in inference. We start by observing that having lateral competition between units is only half a solution. If a unit does not have enough information available to make the right decision, it may be better to not make a decision at all, and give higher layers vague information instead of wrong information.

> **Insight:** Without sufficient information to make a decision, lateral competition might be counterproductive.

The missing factor to perform Bayesian inference is the context-dependent prior, that would tell what result is expected in the situation at hand. The complication is that knowing the current situation requires interpreting the input, leading to a circular dependency:

> "a major problem for vision systems is how to use low-level cues to rapidly access the correct high-level models so as to quickly resolve the low-level ambiguities"[12]

[12] Alan Yuille et al. (2006)

Luckily, with only partial or uncertain interpretations it may already be possible to get a rough idea of the situation, and use the model's knowledge to refine the interpretation further.

Since all units will then directly or indirectly depend on each other, a recurrent network results in which units that face ambiguous input may settle to their final state only after units above it have provided it with enough context to choose the correct interpretation. The network may be quite different from usual recurrent networks, since there is a distinction between ascending and descending connections[13]. This layered topology with functional asymmetry creates a structured form of recurrence, and keeps the idea of having a hierarchy of layers intact.

[13] In the most simple design, a learnt weighted sum of the activations from above would be added to the bias of each unit, thus treating them just like normal inputs from below. The network design could however treat downward and upward connections differently, as is the case when using predictive coding, see below.

> **Insight:** Most contemporary networks are either recurrent or hierarchically structured, but the combination would make a lot of sense.

Conveniently, the top-down connections can be exactly the ones that define the generative model we are trying to invert. In the terminology of VAEs and approximate inference models of section §2.2.2, our inference model q *incorporates* the structure and parameters of p. If the generative model p would have been defined beforehand, these top-down connections need not be changed when learning q. More likely however, we will want to learn the generative model and inference model simultaneously, and having them weaved together into one recurrent network will enable us to use either to provide learning signals for the other, as was the idea in the previous chapter, and will be shortly returned to in the next.

> **Insight:** The generative model could be defined only implicitly, by the top-down connections in inference model.

5.4.1 *Inference dynamics*

When dealing with a generative model with several layers, inference with both bottom-up and top-down signalling gets interesting. In the deep network, the iterative inference procedure operates throughout the hierarchy, with each layer continuously conjecturing likely causes (e.g. h_l) for the input below, generating prior expectations for the layer below ($p\left(h_{l-1}|h_l\right)$), getting a prior from above ($p\left(h_l|h_{l+1}\right)$) and combining prior and likelihood to reinterpret its input from below and update the previously hypothesised causes. With many messages flowing up and down, there is quite some freedom in choosing in which order the iterations proceed. One option is to proceed from the bottom upwards to update each layer while sticking with the previous prior, and then walking down again while updating the priors to prepare for the next upward pass. However, it seems more attractive to parallelise the updates and let each layer update their values simultaneously.

Since the sketched inference process behaves like a dynamical system, relevant questions are to which state it will converge, and whether it actually stabilises at all. When each layer tries to produce the single best (MAP) interpretation, the system may still get stuck in a stable situation where a low level and a high level have settled to a locally optimal state, and finding the global optimum would require both layers to step away from their optimum. Even more interestingly, in pathological situations, in particular when hypotheses with high likelihood have a low prior expectation and vice versa, the system might start oscillating between different states[14]. Both these problems seem strongly related to the chosen greedy approach where each layer produces a single best hypothesis. To prevent this problem, ideally the layers would express their full (conditional) probability distributions, but that may be undoable. A possible solution is to apply particle filtering, which means keeping multiple hypotheses around at each layer and combining the information of two layers by finding the best matching pairs of hypotheses:

[14] Interestingly, the brain appears to alternate between interpretations when given seemingly inconsistent input, for example when showing each eye a different ('binocular rivalry') (Andy Clark, 2013).

> "The only remedy that has been found in the computational literature is not to jump to conclusions but to allow multiple high-probability values for the features or hypotheses to stay alive until longer feedback

loops have had a chance to exert an influence."[15]

[15] Tai Sing Lee and David Mumford (2003). Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448

Insight: Letting each layer search a single best value may hinder finding the best collective interpretation.

Another important question is the speed of convergence of the system. Iterative inference mechanisms may be powerful but can be prohibitively slow. Interestingly, the time required to settle to an interpretation may depend on the complexity of the image, which may sound very natural to humans but is remarkably different from the constant computation time of feed-forward networks. The speed of convergence largely depends on design choices like the form of the generative model and exactly how the layers update their values. One factor that could also help is to start with a good initial hypothesis, which could perhaps be the resulting interpretation of a related datum; the obvious example of this is discussed next.

5.4.2 *Temporal data*

The discussed iterative method of inference should still work if the input is not a static value, but a stream of values that is presented one by one. The activations of the previous time step can then be retained to serve as the initial state for the next input, and inputs be fed in at the same rate at which the activations are updated. Even though this would imply that the input has already changed when the higher levels have settled to a new interpretation, there are two effects at play that should make things work nicely.

Firstly, higher-level layers will represent more abstract features that will accordingly change more slowly. In a video, shapes, shadows and positions may change, but a koala does not turn into a penguin from one frame to the next. This property makes that the previously inferred high-level representations can still be used to provide priors for the next input[16], allowing for a quick input processing unless significant changes occur suddenly. To better understand the method, it may be helpful to regard the network as a configurable feed-forward network, with higher layers tweaking the configuration of the lower layers to adjust them for the current situation.

[16] In fact, the high-level features may be more abstract *because* they are less volatile; see section §4.5.2.

Insight: Higher-level features change more slowly, and the recurrent inference network can be regarded as a feed-forward network that continually reconfigures itself for the current situation.

The second helpful effect is that the network can learn to take the progression of time into account. Rather than a static expectation of the current input, the priors that are sent down will present a prediction of what is expected in the subsequent input.

5.4.3 *The case without feedback*

Stepping back for a moment, we can again ask how networks manage that are unable to read the context information from the layers above. Intuitively, we obtain a situation similar to the case when competition is lacking, discussed earlier: for optimal detection of their pattern, the units will need to try to extract contextual hints from the input below them, thereby conflating information about the prior of the features with information about the likelihood of the features themselves. For example, a unit trying to detect a koala in the center of view may attempt to look whether the area around it looks like a (eucalyptus) tree. Although understandable and possibly helpful for inference, conflating the two types of information at this level seems suboptimal and may hamper generalisation, because it will have to use the lower-level features.

It would be nice if tree-detecting units would be available below the koala-unit, but the fundamental problem here is that there is no logical order in which the features can be resolved: how probably the object is a koala depends on whether it is on a eucalyptus tree, but also the other way around, the treey stuff is more probably a eucalyptus tree if the object on it is a koala. Or, thinking back to the earlier example, each squiggle is more probably a letter when the squiggles around them are letters too. In theory, a feed-forward network could in one layer make a initial interpretation of the squiggles, and then refine those in a subsequent layer by taking the initial interpretations of the surrounding squiggles into account. Such an approach makes a rough attempt at learning to compute what the iterations of the network incorporating feedback would perform, but does so after having unfolded the recurrent network and decoupled the shared weights. Learning this way would be much harder because there are many more weights and they have to be balanced properly.

5.5 *Predictive coding*

The discussed principle of using the generative model to bias competitions between the units below comes close to the top-down processing theories of the brain that were covered in section §3.4. However, one important aspect of those theories that up to now has been ignored in this chapter, is the idea that ascending connections need to convey only that part of the information that was not already present in the layer they report to. Put simply, a lower layer would send up only those activations that do not match with the expectations it receives through the descending connections. Using this method of predictive coding in our inference model would intuitively make sense, because for updating an interpretation, a higher layer mostly needs to know what is *wrong* with its interpretation, and hearing everything it already expected to hear is merely a distraction from that.

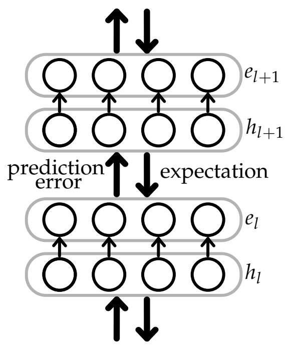

Figure 5.4: Predictive coding introduces error units, which report upwards only the mismatch between the top-down expectation and the actual activations.

Besides biasing the units below, the generated expectations can thus be used to filter the information that is sent up. The network required for this can best be drawn as having two types of units: each usual *representation unit* is hard-wired to an *error unit* (see figure 5.4). Although predictive coding requires a structural change in the network architecture, we will see that it fits nicely into the biased competition framework we have been discussing above, because we had not yet defined exactly *how* a layer updates its hypothesised interpretation; we only stated that it in some way bases on the activations from below and the generated prior expectation from above. Before inspecting how the two concepts fit together in section §5.6, we look at an example and some thoughts about how predictive coding creates a negative feedback loop that results in iterative inference in another way than we discussed before.

5.5.1 *Inference in sparse coding*

It may be insightful to realise that the basic principle of predictive coding appears naturally when straight-forwardly applying gradient ascent optimisation to perform MAP inference in a simple generative model[17], thus without using a separate approximate inference model. To exemplify this, we derive this inference method for sparse coding as defined in section 2.4.2, starting from the definition of its single-layer generative model:

[17] Idea drawn from Aapo Hyvärinen et al. (2009), section 14.3

$$\begin{aligned} p(h) &= \prod_i \frac{\lambda}{2} e^{-\lambda|h[i]|} \\ p(x|h) &= \prod_i \frac{1}{\sqrt{\pi}} e^{-(x-Ah)[i]^2} \end{aligned}$$

Performing MAP inference in this model is done by searching[18]:

[18] See equation 2.2.

$$\hat{h} = \arg\max_h p(h|x) = \arg\max_h \{\log p(x|h) + \log p(h)\}$$

By taking the gradient of the objective in the accolades with respect to h, we find the following rule for updating our hypothesis:

$$
\begin{aligned}
\Delta h &\propto \frac{\partial \log p\left(x|h\right)}{\partial h} + \frac{\partial \log p\left(h\right)}{\partial h} \\
&= \frac{\partial \sum_i -\left(x - Ah\right)[i]^2}{\partial h} - \frac{\partial \sum_i \lambda \left|h[i]\right|}{\partial h} \\
&= A^T\left(x - Ah\right) - \lambda\ \mathrm{sign}(h) \\
&= A^T e - \lambda\ \mathrm{sign}(h)
\end{aligned}
$$

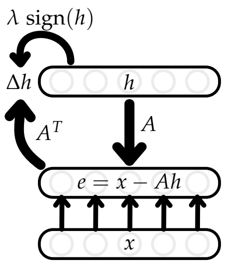

Figure 5.5: The computation for finding the MAP interpretation through gradient ascent.

It can be seen (especially from figure 5.5) that the update consists of two parts, one due to the error $e = x - Ah$ between the generated expectation and the actual input, and one due to the sparsity prior .

It is also easy to see that this property does not only appear for this particular choice of probability distributions, and variations of the generating distribution $p\left(X|h\right)$ may lead to the same network structure but using a different computation for the error $e\left(x, Ah\right)$[19]. For the Gaussian distribution used here, the mismatch between the activation x and the expectation Ah is measured by subtracting them, while other distributions, for example a Bernoulli distribution for binary activations, would lead to other measures for their mismatch.

[19] At least when the generating distribution is factorisable, and $p\left(x[i]|h\right) = f\left(x[i], (Ah)[i]\right)$, for a differentiable function f.

5.5.2 *The power of negative feedback*

The point of the above example is that predictive coding is quite a natural approach to perform inference. An important property of the just derived iterative inference procedure, in contrast to the encoder of an auto-encoder, is that it indirectly creates a competition between alternative explanations. Instead of units competing directly with each other, predictive coding ensures that as soon as the emerging interpretation provides an explanation for some part of the input, that part is muted by the generated expectation and will thereby not cause other explanations to activate.

Insight: Predictive coding creates an indirect competition between explanations by silencing already explained parts of the input.

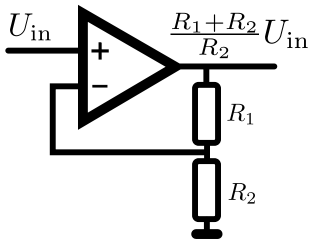

Figure 5.6: Typical example of negative feedback in an electronic circuit.

To understand predictive coding at a more abstract level, it may be illustrative to observe the relation with a negative feedback amplifier in electronics. In such a circuit (see figure), the output of an high-gain amplifier is divided and fed back and subtracted from its input, so that only the difference between the two values is amplified, and the output is stable as soon as those values are equal. Although this approach may seem devious, because it compensates for its own imperfections this type of amplifier is in many ways superior to an amplifier without negative feedback, whose inherent imperfections lead to distortions in the output.

Although the situation may be more complex in neural networks than in most applications of control theory, the important principle is to compensate for imperfections by validating the output and internally making the error available as a signal. The mapping from such an error signal to the desired activation may be much simpler than a direct mapping from the original input to that activation.

Insight: Prediction errors should be made signals, because measuring how wrong the output is eases determining the correct output.

Intuitively, evaluating your mistakes and correcting for them is easier than computing the right answer without looking back[20]. An implied assumption here is that although the mapping the input to the right output is difficult, it is easy to check how far the output is off. In electronics, dividing voltages is much easier than multiplying them, so creating the inverse of an amplifier is a piece of cake. The crucial question is now whether the same criterion holds for applications of machine learning. If it is the case, as has been tacitly assumed throughout this thesis, that an adequate directed generative model of our data could be made, then to perform inference it seems sensible to let generation provide some kind of negative feedback to the inference procedure.

[20] Seen this way, feed-forward computation is like drawing blindfolded.

Insight: If — and only if — validating the output is easier than producing it, a feedback loop may be the solution.

5.6 *Predictive coding + expectation bias*

Having sufficiently evangelised about how predictive coding helps a layer process its inputs from below, it is time to sketch how predictive coding is combined with the earlier idea of using top-down contextual information to decide between interpretations and disambiguate inputs. Although the two principles are often taken together in the brain theories, they are best understood separately. The network in figure 5.4 does send down its expectations at every layer, but still it does not create a continuing top-down path, so activations are not able to use knowledge from the layers above it.

5.6.1 *Combining the two*

When adding in the idea of biased competitions, the expectation received from above is used both to set the prior of the layer, and to filter the information that is sent back up. These two functions fit together very nicely. Intuitively, each representation unit tries to set its activation as to match with the expectation it receives, but if this would not be consistent with the input from below, its deviation from the expected activation is reported upwards.

Insight: Error coding and disambiguation are two complementary uses of top-down feedback.

To implement this idea in a network architecture, the representation units need to receive a signal from above. To provide the biases to the representation units, the error units can conveniently be reused, by adding a connection back to the representation unit they accompany. Biasing then boils down to encouraging each representation unit directly to make the error sent up as small as possible, which allows us to regard the whole network as being solely engaged in prediction error reduction.

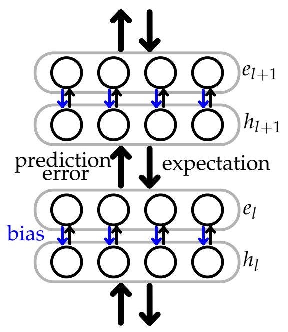

Figure 5.7: A bias makes the representation units try to minimise the error that is sent upwards, thus trying to match the expectation itself.

Insight: When combining predictive coding with expectation bias, the goal of a unit is to minimise the error it receives from below, as well as the error it sends up.

Alternatively, the biasing connections could stem directly from the representation units in the layer above, as was the idea in the previous chapter. This approach provides an equivalent effect[21], but a possible downside is that two downward paths are made that would both have to learn to generate expectations.

[21] Michael Spratling (2008). Reconciling predictive coding and biased competition models of cortical function. *Frontiers in Computational Neuroscience*, 2

5.6.2 *"Deep sparse coding"*

Like in the sparse coding example above, we can see the described network architecture appear naturally by deriving an inference procedure in a generative model. In the single-layer model we studied in that example, the hidden layer prior was fixed and centered around zero, but it is easy to imagine that prior to instead be centered around a value generated by a new layer h_2 that is put on top. For simplicity we could make this prior $p(h_1|h_2)$ Gaussian[22], while the new layer h_2 would have a sparse prior. Inferring the value of h_1 through gradient ascent (assuming h_2 fixed) would in that model lead to the following iterative inference step:

[22] A sparse prior around a non-zero value makes little sense anyway. Note that we fix the variance again to $\sigma^2 = 1/2$ so it disappears from the equations.

$$\begin{aligned}
\Delta h_1 &\propto \frac{\partial \log p\left(x|h_1\right)}{\partial h_1} + \frac{\partial \log p\left(h_1\right)}{\partial h_1} \\
&= \frac{\partial \log p\left(x|h_1\right)}{\partial h_1} + \frac{\partial \log p\left(h_1|h_2\right)}{\partial h_1} \\
&= \frac{\partial \sum_i -(x - A_1 h_1)[i]^2}{\partial h_1} + \frac{\partial \sum_i -(h_1 - A_2 h_2)[i]^2}{\partial h_1} \\
&= A_1^T\left(x - A_1 h_1\right) - \left(h_1 - A_2 h_2\right) \\
&= A_1^T e_0 - e_1
\end{aligned}$$

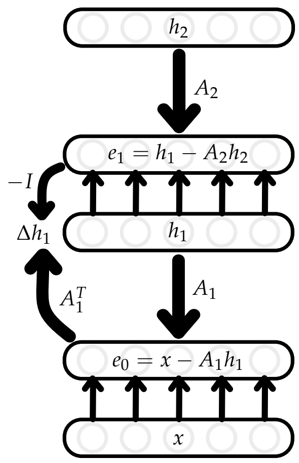

Figure 5.8: Layer h_1 is updated to minimise both the error from above and from below.

When performed at each hidden layer simultaneously, this result matches exactly with the architecture we have been describing, with a layer h_l getting two update signals that try to reduce both the incoming prediction error e_{l-1} from below, and the error e_l between its activation and the received prior expectation from above (unless it is the top layer). In fact, if we add one more layer to it, the simple case shown here turns out to be equivalent to the predictive coding model experimented with by (Rajesh Rao et al., 1999), mentioned earlier in section §3.4.2. Using Gaussian distributions and plain linear generation may however not bring the most interesting models, but choosing different types of generative models can lead to more satisfying models that still have a similar structure.

For instance, we could change the probability distribution to create sparsity in the intermediate layers too, possibly by factorising the prior into a part influenced by top-down generation, and a sparsity prior: $p\left(h_1|h_2\right) = p_{td}\left(h_1|h_2\right) p_s\left(h_1\right)$. In a way, the kind of model we are looking for here could be seen as a logical extension of sparse coding to contain multiple layers. One more important aspect that would be helpful for creating powerful models is to let layers not only generate the particular value that they expect to see, but instead enable them to express a whole range of values that would be expectable. This is the next, and last, topic we look at in this chapter.

5.7 *Heteroscedasticity*

In the above, our generative model would always generate a single expected value for each unit in the layer below, and a unit tries to take a single value close to its received expectation. How far it can deviate from that expected value was fixed by the hard-coded variance of the chosen distribution. Because many types of data are heteroschedastic, i.e., the variance of variables depends on other latent variable values, it would be much better if the model can also express levels of uncertainty, for example by individually computing the variance of the prior distribution generated for each unit.

5.7.1 *Uncertainty in expectations*

The idea about generating not a single expected value but a wider distribution of expectable values is also explained by David Mumford (see section §3.4.1), who phrased the argument as "templates must be flexible"[23]. Such flexibility gives a new type of power to the generative model. Imagine we are trying to generate our expectation for the colour of an object, where we for now assume the value of some single unit represents this colour. When the system is able to assign a variance, or another measure of uncertainty, to its computed expected value, for different animals we can have different amounts of certainty about their colour, so expectations of a value can be made precise (e.g. koalas are always grey/brownish), or can be deliberately less specific (e.g. boats can be of any colour between red and blue).

[23] David Mumford (1992). On the computational architecture of the neocortex: II. The role of cortico-cortical loops. *Biological cybernetics*, 66(3):241–251

When assuming Gaussian distributions with fixed variance, as is quite commonly done for simplicity, the model is unable to model uncertainty[24]. The generative model then has no way to express indifference about some units of a layer, so it always has to specify an aspect in the same level of detail for every object. When using predictive coding, each unit will always try to fit its value to that expectation, and error units will keep reporting mismatches needlessly. The layer above will therefore have to represent the irrelevant details, like the colour of a boat, in order to shush the reported errors.[25] When given the ability to model variance, assigning a large variance to the generated expectation for some representation unit practically means that the gain of the corresponding error unit is lowered, so that both the reported error is diminished and the representation unit is pressured less to be close to the expected value.

[24] Though, if not using (batch) normalisation, the model could perhaps play a bit with the scale of values to adjust the relative amount of the variance.

[25] This reminds of the problem described in section §1.5, about the layers having trouble with dropping details.

Insight: A generative model should be able to model not only values but also variances, so it can choose which aspects it is fussy or indifferent about.

5.7.2 *Uncertainty in representations*

Related to the ability to generate flexible expectations, a similarly useful power would be the ability to model the uncertainty of interpretations, which would be especially useful during an iterative inference process. As discussed in section §5.4.1, when sending signals up and down to combine evidence and knowledge, it would be helpful to keep multiple hypotheses alive in order to not get stuck in a local optimum. The idea of uncertain interpretations is a step towards this idea, because if an interpretation can express uncertainty about its value, it becomes more like a range of hypotheses instead of a single best guess.

In an inference process, the hypothesis of a layer may initially be very uncertain, so it would generate and send down a prior that also contains uncertainty, thus telling the layer below that the context is unknown and pretty much anything is expected. The (unproven) intuition is that after a few iterations, the flow of signals up and down makes that the hypotheses of layers become more specific, and if all works out well they should converge towards the MAP value.

Insight: Modelling uncertainty in interpretations allows inference to settle from a clueless hypothesis to a sharp value.

Another interesting idea that combines predictive coding with the ability of setting the variance of expectations, is that an attention mechanism could be implemented that can turn its focus to some aspect of an input by assigning its expectations for that aspect a very small variance. For the units that are attended, even small mismatches from the expected value will then be reported upwards as errors[26]. This principle has been suggested as a possible mechanism to explain attention in the brain:

[26] To be able to focus on an aspect without forcing our expectation onto it, perhaps predictive coding and expectation bias should then not share the same error unit, as was opted in section §5.6.1

"attention might not be the 'selection' of sensory channels but an emergent property of 'prediction'; where high-precision prediction-errors enjoy greater gain."[27]

[27] Karl Friston (2009). The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7):293–301

5.7.3 Sparse representations as probability distributions

To be able to model not only values but also variances of both predictions and representations, we do need to decide on an architecture that has these capabilities. A possible solution is to use sparse binary representations, which not only enables us to model variance, but gets even a step closer to the more powerful capability of expressing arbitrary probability distributions.

The idea of sparse binary representations is that every would-be continuous variable is now discretised, and every possible value it could take is represented by a separate unit that competes for activation with the others, not unlike in a one-hot encoding. This approach of using a separate unit for each value allows to express uncertainty in representations, because several units can be turned on at once if the exact value is not known[28]. Although the representation is made of binary variables, the units that implement it are still continuous, and their level of activation now corresponds to the probability of their binary variable being turned on.

[28] To allow multiple values to be active, the competition should not *enforce* the one-hot encoding. In contrast, we could regard a continuous-valued unit as holding a super-strict competition between its possible values, because only one out of many possible values can be the outcome.

Insight: Sparse representations are very suitable for expressing probability distributions.

Sparse representations likewise support generating flexible expectations. Instead of supplying an expected value and its variance to some continuous variable, the top-down signal now tells for each unit whether its value is within the expected range or not, or (between those two options) exactly how (un)expected their activation would be[29]. In order to make this idea work, we do need to change the way we treat about the expected value sent to a unit. In the predictive coding approaches covered so far, we would have each unit report its deviation from the expected value, possibly divided by the expectation's variance. However, in the new approach, a generated expectation reports to a range of units that *any* of them is expected to be active, but not *all* of them, so units that fall within the expected range but are not active should not start reporting an error because their inactivation does not match the received expectation. In other words:

[29] To prevent massive over-fitting, perhaps we are able to use excitatory lateral connections to by default group multiple values.

Insight: In sparse representations we need 'asymmetric' error reporting, to report unexpected activity but not unexpected *in*activity.

In terms of probability distributions, this could be achieved by letting the prior distribution that is sent down to a unit not be centered around the computed (expected) value, but to range from zero up to that value, so that a weaker activation is acceptable too. In an implementation, things may be as simple as letting the top-down signals inhibit error units when their activation is expected. To still be able to require a unit to be inactive, a solution could be to add units with complementary activations.

6

Synthesis

In the course of this thesis, we have studied how to form abstract data representations by learning directed generative models (chapter 1), reviewed typical neural network algorithms that implement this way of learning (chapter 2), and drawn inspiration from theories about bidirectional signalling in the brain (chapter 3) in order to investigate alternative ways to learn network weights (chapter 4), as well as ways to more accurately infer the representation of given data (chapter 5). To round off this thesis, we look how these explored ways of learning and inference fit together, to briefly sketch the outline of a network architecture and learning method that could be used to guide further research and experiments.

6.1 Unifying learning and inference

In chapter 4, the line of thought was that instead of accompanying both computational paths (inference and generation) with an opposing path through which gradients are back-propagated from an objective defined at the very end, we could create learning rules that determine helpful weight updates based on the information that is made locally available via the already existing paths. The inference model listens to the feedback it receives from the generative model, and vice versa, to determine how it can be more useful. From the ideas of target propagation and recirculation, a useful perspective we discovered was to regard the generative model to be reusing the same units that the inference model uses, together forming a recurrent network in which a unit's weight update can be derived simply by comparing its activation and inputs before and after the influence of top-down feedback. If the unit's new values would come from the actual Bayesian posterior of the generative model, they would be ideal to use as target outputs for the units' input.

This last idea fits together remarkably well with the idea of chapter 5, in which the goal was to make the inference model more capable of approximating Bayesian inference by incorporating the generative model into the inference network architecture. Each layer would generate expectations to change the bias for units below and filter out already expected activity, thereby creating feedback loops and turning inference into an iterative process that gradually forms its best interpretation.

These approaches to learning and to inference look like they were made for each other[1]: For local learning from temporal differences in activation, we wanted the feedback loop to give each unit a justifiably better value that can be used as its target value, and iterative inference provides exactly that. The other way around, iterative inference would be very hard to train by back-propagating through all iterations, and learning by local update rules may be a possible solution because the feedback loops could already provide all information that is needed for choosing a good update direction.

[1] This of course is the reason both are treated in the same thesis.

The combination of the ideas boils down to an algorithm that, given a datum, forms an initial interpretation in a feed-forward pass, then starts using the feedback loop to refine the interpretation and settle to an optimum, after which the weights are adjusted so that for a similar future input, the initial interpretation will already be closer to the ultimate refined value, and inference will be expedited. The paradigm can be considered an amortised inference approach in which the network provides its own truth value, so that next time it will converge faster to whatever value it will converge to now. Although this may sound circular and potentially only worsening its weights, the hypothesis here is that by incorporating the generative model in the inference network and hard-wiring the predictive coding principle into its architecture, learning by using future values as targets will work out fine.

Insight: In a network that gradually improves its interpretation, the future value of a unit is a good target value for its current input.

Another important point to take into account here is that the generative model is not static or predefined. Instead of saying that the inference network incorporates the generative model, we could say that its top-down connections *define* it. The generative model is learnt simultaneously with the inference network, and it can be learnt in pretty much the same way: having inferred the optimal interpretation, the probability to generate the current input below from that interpretation should be increased. We could also look at it from the perspective of the inference process, and say that we update the weights such that the current input should be expected in the currently represented situation.

The just described way of learning is similar to the normal sparse coding learning algorithm described in section §2.4, but now performed at every layer in the deep network, and is even more similar to the learning method used in the predictive coding algorithm of Rajesh Rao and Dana Ballard[2]. In the predictive coding architecture, the learning rules for both types of units turn out to be simple and (anti-)Hebbian, which gives them an elegant, intuitive explanation:

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

Insight: Representation units learn to predict their own state of activation and try to reach it faster next time. Error units learn to predict their own activation from signals from above, and try to prevent themselves from firing in the future.

6.2 Further research

The generality with which the methods and ideas have been described in this thesis makes that getting to a concrete implementation still requires a lot of work, and it may take many trials to find a working approach among the many possible manifestations. A rough outline of the type of architecture has been sketched, but there is still a huge amount of freedom regarding which pieces and principles to select and how to configure them. There are too many aspects that need further thinking and experimentation to list them all. Some exploratory experiments that may be worth a try could be done by digging up some of the previous work that has been referred to, and look if tweaking their approaches leads anywhere. For example, the predictive coding implementation of Rajesh Rao et al. (1999) could be an interesting place to start.

Although the idea in this thesis is that the learning algorithm would best be designed along with the inference procedure, the principles of some architectural aspects could already be tested with generic optimisation methods. A few possible experiments that come to mind are to create learnable lateral competition in auto-encoder inference networks, to model variance in methods that normally use fixed-variance Gaussians, and to try what happens when a hierarchical recurrent network with both ascending and descending connections is trained (with back-propagation through time) to produce the true posterior of a known generative model. It would be interesting to see if the latter would learn to perform predictive coding.

Besides trying to work out the principles that have been described, an important task is to question the principles themselves. Many assumptions have been made, and discovery of their falsehood would undermine the conclusions that have been drawn. One such assumption used throughout this thesis is that we want to learn a directed generative model, while perhaps that might not even be so useful. We assumed we want to perform inference in this generative model in order to interpret inputs. It could be questioned whether this is really that important. Bayesian inference is not a goal on itself and may be an intractably difficult task, while for any kind of intelligent system we may desire to build, what we would actually need is Bayesian decision making: a brain or computer needs to extract information from its surroundings that is relevant for deciding its actions, and inference is only useful in so far as it is required for making decisions. Another question is to what extent the concepts are capable of performing more abstract information processing[3], where attention and more complex reasoning mechanisms are required. Probably at some level the hierarchy of abstraction that makes sense for the interpretation of low-level data has to make place for a less structured mesh of associations, like it also seems to be the case in the correspondingly named association areas of the neocortex.

[3] Maybe *thinking* is the right word here

6.3 Conclusion

The main goal of this thesis was to explore ways to improve unsupervised deep learning, while drawing inspiration from theories about the brain and looking at existing and past work in this direction that could be worth revisiting. Many well thought-through ideas have been proposed and tried over the last few decades, and some may lead to successful results when properly implemented and mixed into modern methods and technology.

The general direction has been to move away from feed-forward networks trained with end-to-end back-propagation. Instead, the intention is to move towards methods that could be considered a deep sparse coding structure, with learnt iterative inference procedures that are able to approximate Bayesian inference, while locally available information is used to at each layer to improve the generative model and learn to speed up the inference process.

Perhaps more important than the particular techniques that have been suggested, is the way of thinking about them. The implicitly advocated way of doing research is to reason about the algorithm design and study fundamental principles of learning, rather than to tweak some parts of a well-performing algorithm and hoping to beat a benchmark score by a few permille. Quite some research, including that in academia, seems to have a tendency towards the latter approach, but while we are still in the stone age of machine learning, focussing on accuracy rates feels like premature optimisation.

Insight: To take the field forward, it may help to take a step back. Instead of feed-forward, we should focus on feed*back*.

7

Bibliography

Guillaume Alain and Yoshua Bengio (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593.

Yoshua Bengio (2013). Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pages 1–37. Springer. 17

Yoshua Bengio (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv:1407.7906*. 53, 55

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov (2015). Importance weighted autoencoders. *arXiv:1509.00519*. 31

Santiago Ramón y Cajal (1899). *Comparative study of the sensory areas of the human cortex*. 34

Kyunghyun Cho (2013). Simple sparsification improves sparse denoising autoencoders in denoising highly corrupted images. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 432–440.

Kyunghyun Cho (2014). *Foundations and Advances in Deep Learning*. PhD thesis, Aalto University.

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220. 48

Andy Clark (2013). Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(03):181–204. 68

Andy Clark (2014). Bayesian predictive coding. Recorded talk in a Greenland trip organised by the Moscow Center for Consciousness Studies. 39

Thomas Cleland and Christiane Linster (2012). On-center/inhibitory-surround decorrelation via intraglomerular inhibition in the olfactory bulb glomerular layer. *Frontiers in integrative neuroscience*, 6.

Arthur Dempster, Nan Laird, and Donald Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Daniel Felleman and David Van Essen (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47. 35, 36

Karl Friston (2005). A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836.

Karl Friston (2009). The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7):293–301.

Wulfram Gerstner and Werner Kistler (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press. 55

Ian Goodfellow (2015). Do statistical models 'understand' the world? Talk at the RE.WORK deep learning summit 2015.

Daniel Graham and David Field (2006). Sparse coding in the neocortex. *Evolution of nervous systems*, 3:181–187. 16

Karol Gregor and Yann LeCun (2010). Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 399–406.

Stephen Grossberg (1980). How does a brain build a cognitive code. *Psychological Review*, pages 1–51. 38

Stephen Grossberg (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63.

Kenneth Harris and Thomas Mrsic-Flogel (2013). Cortical connectivity and sensory coding. *Nature*, 503(7474):51–58.

Jeff Hawkins and Sandra Blakeslee (2007). *On intelligence*. Macmillan.

Hermann von Helmholtz (1867). *Handbuch der physiologischen Optik*. Voss.

Geoffrey Hinton (2000). Modeling high-dimensional data by combining simple experts. In *AAAI/IAAI*, pages 1159–1164. 64

Geoffrey Hinton (2007a). How to do backpropagation in a brain. Invited talk at the NIPS 2007 Deep Learning Workshop.

Geoffrey Hinton (2007b). To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547. 18

Geoffrey Hinton and James McClelland (1988). Learning representations by recirculation. In *Neural information processing systems*, pages 358–366. New York: American Institute of Physics.

Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554. 52

Geoffrey E Hinton (2007). Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434.

Jonathan Horton and Daniel Adams (2005). The cortical column: a structure without a function. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):837–862.

Yanping Huang and Rajesh Rao (2011). Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593.

David Hubel and Torsten Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106.

Heikki Hyötyniemi (1996). Turing machines are recurrent neural networks. *Proceedings of STeP*, pages 13–24. 59

Aapo Hyvärinen, Jarmo Hurri, and Patrick Hoyer (2009). *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.*, volume 39. Springer Science & Business Media. 22, 71

Ulugbek S Kamilov and Hassan Mansour (2015). Learning optimal nonlinearities for iterative thresholding algorithms. *arXiv:1512.04754*.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun (2010). Fast inference in sparse coding algorithms with applications to object recognition. *arXiv:1010.3467*.

Diederik Kingma and Max Welling (2013). Auto-encoding variational bayes. *arXiv:1312.6114*. 31

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 48

Yann Le Cun (1986). Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer. 52

Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng (2011). Building high-level features using large scale unsupervised learning. *arXiv:1112.6209*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015). Deep learning. *Nature*, 521(7553):436–444.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio (2015). Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer.

Tai Sing Lee and David Mumford (2003). Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448.

Qianli Liao, Joel Leibo, and Tomaso Poggio (2015). How important is weight symmetry in backpropagation? *arXiv:1510.05067*.

Timothy Lillicrap, Daniel Cownden, Douglas Tweed, and Colin Akerman (2014). Random feedback weights support learning in deep neural networks. *arXiv:1411.0247*. 51

David MacKay (1995). Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, Section A*, 354(1):73–80.

Alireza Makhzani and Brendan Frey (2013). k-sparse autoencoders. *arXiv:1312.5663*.

Henry Markram, Maria Toledo-Rodriguez, Yun Wang, Anirudh Gupta, Gilad Silberberg, and Caizhi Wu (2004). Interneurons of the neocortical inhibitory system. *Nature Reviews Neuroscience*, 5(10):793–807. 34

David Mumford (1991). On the computational architecture of the neocortex: I. The role of the thalamo-cortical loop. *Biological cybernetics*, 65(2):135–145.

David Mumford (1992). On the computational architecture of the neocortex: II. The role of cortico-cortical loops. *Biological cybernetics*, 66(3):241–251. 36, 38

Bruno Olshausen et al. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609.

Randall O'Reilly (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938.

Rajesh Rao and Dana Ballard (1997). Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural Computation*, 9(4):721–763.

Rajesh Rao and Dana Ballard (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87. 75, 81

Brendan van Rooyen and Robert Williamson (2015). A theory of feature learning. *arXiv:1504.00083*. 16

Jürgen Schmidhuber (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Jitendra Sharma, Alessandra Angelucci, and Mriganka Sur (2000). Induction of visual orientation modules in auditory cortex. *Nature*, 404(6780):841–847.

S. Murray Sherman and R. W. Guillery (2002). The role of the thalamus in the flow of information to the cortex. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 357(1428):1695–1708.

Stewart Shipp (2007). Structure and function of the cerebral cortex. *Current Biology*, 17(12):R443–R449. 36

Michael Spratling (2008). Reconciling predictive coding and biased competition models of cortical function. *Frontiers in Computational Neuroscience*, 2.

Rupesh Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber (2013). Compete to compute. In *Advances in Neural Information Processing Systems*, pages 2310–2318.

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

Harri Valpola (2014). From neural PCA to deep unsupervised learning. *arXiv:1411.7783*.

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

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

Catherine Wacongne, Etienne Labyt, Virginie van Wassenhove, Tristan Bekinschtein, Lionel Naccache, and Stanislas Dehaene (2011). Evidence for a hierarchy of predictions and prediction errors in human cortex. *Proceedings of the National Academy of Sciences*, 108(51):20754–20759.

Wikipedia (2015). Brodmann area — Wikipedia, the free encyclopedia.

Xiaohui Xie and Sebastian Seung (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural computation*, 15(2):441–454.

Alan Yuille and Daniel Kersten (2006). Vision as bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308. 67