Chapter 1

Deep Learning of Brain Images and its Application to Multiple Sclerosis

1.1 Introduction

What is deep learning? Deep learning is used to describe a multitude of learning-based methods with certain common characteristics: a) the use of multiple layers of nonlinear processing units, and b) the unsupervised or supervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features. In this chapter, we will introduce the most commonly used deep learning methods for medical image analysis. We start with a description of unsupervised models like the restricted Boltzmann Machine (RBM), which are the building blocks of deep belief networks, a model that can be used for learning a hierarchical set of features from input images without the need of labels. Later, we will introduce supervised models like neural networks, how these models can be adapted for unsupervised learning, e.g., stacked auto encoders, and scaled to larger images using convolutional neural networks.

The distinction between supervised and unsupervised models is lose. Primarily unsupervised models can also be used for supervised tasks like classification, while supervised models can be used for unsupervised feature learning, by reformulating the unsupervised problem to a supervised problem. Examples will be given below.

1.1.1 Deep Graphical Models

Unsupervised models are mostly used for the automatic learning of features of an image domain from a representative data set. They are particularly interesting for medical image analysis as they can be trained without the need of

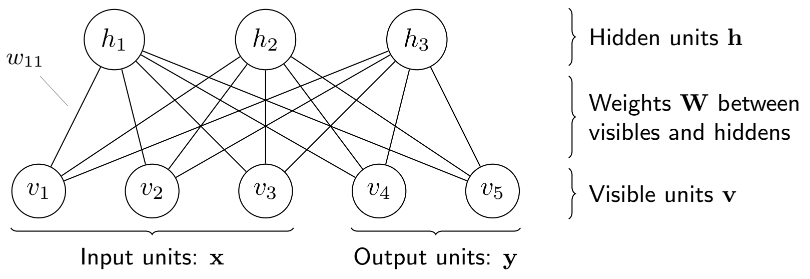
w_{11}
h_1
h_2
h_3
Hidden units $\mathbf{h}$
Weights $\mathbf{W}$ between visibles and hiddens
v_1
v_2
v_3
v_4
v_5
Visible units $\mathbf{v}$
Input units: $\mathbf{x}$
Output units: $\mathbf{y}$

Figure 1.1: Graph representation of an RBM with 3 hidden and 5 visible units. An RBM is used to model the joint probability of input, output and hidden units. Output units are optional. Edges between vertices denote conditional dependence between the according random variables.

labelled data (e.g., segmented images, images with patient data), which can be difficult to obtain. In this chapter, we will first introduce the restricted Boltzmann machines, which are the building blocks of the later described deep belief networks.

Restricted Boltzmann Machines

A restricted Boltzmann machine is a probabilistic graphical models defined by a bipartite graph as shown in Figure 1.1. The units of the RBM are binary random variables. They are divided into visible units $\mathbf{v}$ and hidden units $\mathbf{h}$. There are no direct connections between units of the visible or hidden layer. The RBM defines the joint probability of visible and hidden units in terms of the energy E which is given by:

$$p(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})} \tag{1.1}$$

with

$$-E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_{i,j} v_i w_{ij} h_j + \sum_i b_i v_i + \sum_j c_j h_j \tag{1.2}$$

$$= \mathbf{v}^{\mathrm{T}}\mathbf{W}\mathbf{h} + \mathbf{b}^{\mathrm{T}}\mathbf{v} + \mathbf{c}^{\mathrm{T}}\mathbf{h} \tag{1.3}$$

and $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$.

Inference Inference is the basis for most calculations of an RBM (e.g. training, sampling, encoding, decoding, ...). Since the hidden units are conditionally independent given the visible units and vice versa, inference can be calculate efficiently in an RBM. The posteriors are given by:

$$p(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta}) = \mathrm{sigm}(\mathbf{w}_{\cdot,j}^{\mathrm{T}}\mathbf{v} + c_j) \tag{1.4}$$

$$p(v_i = 1 \mid \mathbf{h}, \boldsymbol{\theta}) = \mathrm{sigm}(\mathbf{w}_{i,\cdot}^{\mathrm{T}}\mathbf{h} + b_i) \tag{1.5}$$

where $\mathrm{sigm}(x)$ is the sigmoid function defined as $\mathrm{sigm}(x) = (1+\exp(-x))^{-1}, x \in \mathbb{R}$.

Sampling Important expectations used for classification, $\mathbb{E}[\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}]$, or training, $\mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \boldsymbol{\theta}]$, can not be calculated directly. In this case, they need to be approximated by the average of samples drawn from the according models. Sampling from the posteriors forms the basis for generating samples from other models. The posteriors can be sampled using 1.4 and 1.5, respectively:

$$h_j = \mathbb{I}(y_j < \mathbb{E}[h_j \mid \mathbf{v}, \boldsymbol{\theta}]) \qquad \text{with } y_j \sim \mathrm{U}(0,1) \tag{1.6}$$

$$v_i = \mathbb{I}(x_i < \mathbb{E}[v_i \mid \mathbf{h}, \boldsymbol{\theta}]) \qquad \text{with } x_i \sim \mathrm{U}(0,1) \tag{1.7}$$

where $z \sim \mathrm{U}(0,1)$ denotes a sample from the uniform distribution in the interval $[0,1]$ and $\mathbb{I}$ is the indicator function, which is defined as 1 if the argument is true and 0 otherwise.

A sample from the joint probability, $p(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$, can be drawn using block Gibbs sampling. For this purpose, $\mathbf{v}$ is initialized with random values at the beginning. Then, the two posteriors $p(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta})$ and $p(\mathbf{v} \mid \mathbf{h}, \boldsymbol{\theta})$ are sampled alternately. If this chain is run for sufficiently many iterations, samples drawn from the posteriors are distributed according to $p(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$.

Learning RBMs can be trained using maximum likelihood estimation (MLE). The gradient of the log likelihood function is given by:

$$\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D} \mid \boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] - \mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \boldsymbol{\theta}] \tag{1.8}$$

The first expectation can be estimated using a mean field approximation:

$$\mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] \approx \mathbb{E}[\mathbf{v} \mid \mathbf{v}_n, \boldsymbol{\theta}]\mathbb{E}[\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] \tag{1.9}$$

$$= \mathbf{v}_n \mathbb{E}[\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] \tag{1.10}$$

The second expectation needs to be estimated using a Monte Carlo approximation:

$$\mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \boldsymbol{\theta}] \approx \frac{1}{S} \sum_{s=1}^{S} \mathbf{v}_s \mathbf{h}_s^{\mathrm{T}} \tag{1.11}$$

where S is the number of samples, and $\mathbf{v}_s$ and $\mathbf{h}_s$ are samples drawn from $p(\mathbf{v} \mid \boldsymbol{\theta})$ and $p(\mathbf{h} \mid \boldsymbol{\theta})$, respectively. If the Gibbs sampler is initialized at a data point from the training set and only 1 Monte Carlo sample is used to approximate the second expectation, the learning rule is also known as contrastive divergence (CD) [**?**]. To speed up learning, the data set is divided into small subsets called mini batches and a gradient step is performed for each mini batch. To avoid confusion with a gradient step, the term "iteration" is generally avoided and the term "epoch" is used instead to indicate a sweep through the entire data set. Additional tricks to monitor and speed up the training of an RBM were summarized by Hinton et al. [**?**].

Gaussian–Bernoulli RBM To model real-valued inputs like the intensities of medical images. This changes the joint probability to:

$$E = bla \tag{1.12}$$

If the visible units are zero mean and unit variance, inference goes like this:

$$\mathbb{E}[v] = bla \tag{1.13}$$
$$\mathbb{E}[h] = bla \tag{1.14}$$

Rectified linear units Hidden state is only binary in an RBM. Encode signal strength by repetition of the same unit with different bias. It can be shown that this type of unit as an activation of exp(...), which can be approximated as $\max(0, x)$. This units are refered to as rectified linear units and have shown to improve classification performance.

Convolutional RBMs To scale to larger images, can use weight sharing and limited receptive field. Can be expressed as a convolution. Greatly reduces the number of trainable weights, employs translational equivariance. Allows RBMs to be trained on high-resolution images.

Training can be further optimized by training in the frequency domain as has been done by some people: ICLR paper, our neural computation paper, fbfft paper.

Deep Belief Networks

RBMs are arguably not deep. To learn a hierarchy of features, RBMs can be stacked on top of each other to form a deep belief network. DBNs are trained layer by layer and are often fine-tuned using a neural network.

For hierarchical feature learning Train on input images. Can be done with convolutional and dense DBNs. Learns a hierarchy of features.

For multi-model feature learning Different pathways that are then later joint.

For classification A special case of multi-modal feature learning, where the output labels are modelled as an additional modality.

1.1.2 Deep Neural Networks

First, we will introduce dense and convolutional neural networks separately. Later we will talk about specific variations, which apply to both models.

Dense Neural Networks

A neural network is a deterministic function. Successive propagation of a signal through multiple layers. Give the equation of a neural network. Neural networks are used to predict the output given some inputs and are trained supervised. Given an error function, we can formulate learning as minimizing the error, as opposed to maximizing the log-likelihood. Optimization can be performed using stochastic gradient descent (SGD). The gradient can be calculated as follows:

- Calculate derivative of error function using the chain rule
- Calculate derivative of the non-linearity using the chain rule
- Calculate derivative of total activation using chain rule
- Proceed to lower layers using the chain rule

This procedure is called backpropagation.

Convolutional Neural Networks

Very similar to dense models but uses a convolution to calculate activation. Show equations for convolutional models and derivative of the gradient using back-propagation.

Training

Training algorithms can be roughly categorized into algorithms that use second order information and algorithms that only use first-order information.

Vanilla stochastic gradient descent is prone to oscillation. This can be limited by using a momentum term. Momentum update favours persistent directions, as they are accumulated.

Nesterov momentum is a slightly different version. Since we gonna take a step in the momentum direction anyways, we calculate the gradient of the look-ahead step instead of the current step. With some massaging to the equations, we can the following update rule.

The exists also a large buddy of work on per-parameter update methods. For example AdaGrad, AdaDelta, Adam, vSGD, eSGD, RMSprop.

There are also a lot of methods that leverage second order information, e.g., Hessian-free optimization used information from the Hessian without ever calculating the full Hessian in order to better navigate pathological curvature. Recenter, it was hypothesised that characteristic points that were believed to be local minima are in fact saddle points. To escape saddle points and to improve optimization, Bengio et al. have developed the so-called saddle-free Newton's method that uses second order information to quickly escape saddle points.

Neural Networks for Unsupervised Feature Learning: Stacked Auto-Encoders

Neural networks can be used for feature learning even when no labels are present. This can be done by using auto-encoder networks. An autoencoder is a neural network with a bottleneck layers, that is trained to predict the input images themselves. The bottleneck layer is used to force the network to learn more abstract features. Auto-encoders can be pre-trained layer-by-layer similar to DBNs. The first auto-encoder has 2 layers and is trained on the training images. Subsequent autoencoders are trained on the hidden activations of the previous auto encoder. I need an image for that. For fine-tuning all layers can be combined into a single network.