Advanced course in machine learning 582744 Lecture 12

Arto Klami

Registered?

Some of you have not registered for the course, so your exercise points are not in TIKLI

If you can register now, do so and send me email so that I know to enter the missing exercise points

If you do not have study permission yet, talk to me after the lecture – you can still take part in the exam, but will get the credits later on

Unsupervised deep learning

The previous examples all considered supervised learning, but neural networks and deep learning match well also unsupervised tasks

Some interesting connections between unsupervised and supervised:

- Unsupervised techniques can be used for *pre-training* of supervised models
- Representations learnt in supervised fashion can be useful for unsupervised tasks
- Autoencoders can be trained using supervised techniques
- Unsupervised models trained on $\mathbf{v} = [\mathbf{x}, \mathbf{y}]$ learn to solve supervised tasks; at prediction time we only observe $\mathbf{x}$

Autoencoders

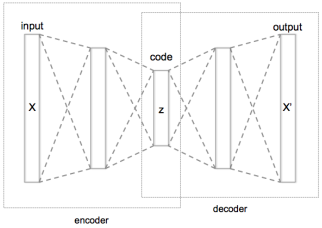
Remember the PCA formulation as lower-dimensional projection that has low *re-construction error*

It is a special case of more general concept of *autoencoder*, a network that learns how to best re-construct the inputs

The information needs to flow through a bottleneck of some sort – otherwise we could just copy the inputs as such. For PCA we keep a subset of the components to constrain the model

Autoencoders

Any neural network trained using the inputs also as outputs is an autoencoder; we can here use supervised training algorithms



Often the first part of the network that compresses the inputs somehow is called *encoder* and the second part is *decoder*, whereas the narrowest part of the representation is the *code* – this matches the compression terminology

We are often not even interested in the outputs; the code is the primary output

Denoising autoencoders

A neat trick is to train autoencoders so that we feed in noisy versions of the vectors but still use the originals as outputs (kind of like the modified inputs in supervised learning)

The network learns to de-noise the inputs and often solves the original autoencoding task more accurately as well

With denoising autoencoders we can actually use codes that are larger than the input – the same would hold if we regularize the network enough

These overcomplete codes correspond to sparse feature extractors

Sequence-to-sequence autoencoders

What if we use recurrent networks as encoders and decoders?

The input and output are sequences of arbitrary length (running text etc) but the representation is of fixed dimensionality

State-of-the-art machine translation is done largely this way, though the output is in another language

Deep generative models

Generative models describe a story that generated the data, and we can naturally write deep generative models

Trained to maximize the log-likelihood of the observations, to model the data density

Ideally we would model the marginal density $p(\mathbf{x})$, marginalizing over the hidden layers

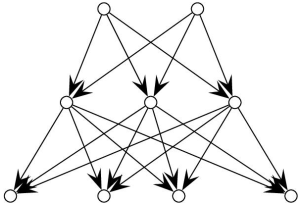
Sigmoid belief network

Sigmoid belief network is a direct formulation for MLP-style generative model: Fully connected network where the weighted sums are passed through sigmoid functions

As a generative model this defines the density

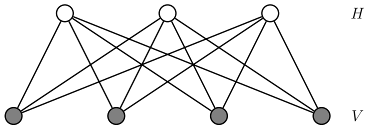
$$p(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) = \prod_i \mathrm{Ber}(v_i | \mathrm{sigm}(\mathbf{h}_1^T \mathbf{W}_{1i})) \prod_j \mathrm{Ber}(h_{1j} | \mathrm{sigm}(\mathbf{h}_2^T \mathbf{W}_{2j})) \prod_k \mathrm{Ber}(h_{2k} | \mathrm{sigm}(\mathbf{h}_3^T \mathbf{W}_{3k})) \prod_l \mathrm{Ber}(h_{3l} | \mathbf{W}_{4l})$$

Introduced already in 1992 but not very widely used; inference is simply too difficult



Restricted Boltzmann machines

A more practical deep generative model is obtained by considering undirected graphs instead



(A restricted) Botzmann machine is defined via pairwise potential energies of (here) binary variables, factorizing the joint density as

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \prod_r \prod_k \psi_{rk}(v_r, h_k)$$

normalized to a density using the *partition function*

$$Z(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \prod_r \prod_k \psi(v_r, h_k)$$

Restricted Boltzmann machines

The factors $\psi(v, h)$ are typically expressed in terms of *energy E* such that

$$p(\mathbf{v}, \mathbf{h}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})} e^{-E(\mathbf{v},\mathbf{h})}$$

with

$$E(\mathbf{v}, \mathbf{h}) = -\sum_r \sum_k v_r h_k W_{rk} = -\mathbf{v}^T \mathbf{W} \mathbf{h}$$

Then the conditional posteriors are easy since the nodes are independent

$$p(\mathbf{h}|\mathbf{v}, \mathbf{W}) = \prod_k \mathrm{Ber}(h_k|\mathrm{sigm}(\mathbf{w}_{:,k}^T \mathbf{v}))$$
$$p(\mathbf{v}|\mathbf{h}, \mathbf{W}) = \prod_r \mathrm{Ber}(v_r|\mathrm{sigm}(\mathbf{w}_{r,:}^T \mathbf{h}))$$

We can interpret $\mathbf{W}$ as *generative weights* and $\mathbf{W}^T$ as *recognition weights* – note the analogy to classical PCA

Restricted Boltzmann machines

Training by the maximum likelihood:

$$\log p(\mathbf{v}, \mathbf{h}|\mathbf{W}) = \mathbf{v}^T\mathbf{W}\mathbf{h} - \log \sum_{\hat{\mathbf{v}}} \sum_{\hat{\mathbf{h}}} e^{\hat{\mathbf{v}}^T\mathbf{W}\hat{\mathbf{h}}}$$

...and as usual we should take expectation over the latent variables to get

$$\mathbf{v}^T\mathbf{W}\mathbb{E}_{p(\mathbf{h}|\mathbf{v},\mathbf{W}_{t-1})}[\mathbf{h}] - \log Z(\mathbf{W})$$

The derivative of the first term wrt to W_{rk} is simply $v_r\mathbb{E}_{p(h_k|v_r)}[h_k]$, and we know the conditional expectation

The latter term can be simplified as well, but it is an expectation that has to be summed over 2^R2^K terms

$$\frac{\partial \log Z}{\partial W_{rk}} = \sum_{\hat{\mathbf{v}}} \sum_{\hat{\mathbf{h}}} p(\hat{\mathbf{v}}, \hat{\mathbf{h}})\hat{v}_r\hat{h}_k$$

Restricted Boltzmann machines

A practical training algorithm uses *contrastive divergence* to approximate the gradients

The basic idea is to compute the latter expectation by sampling:

- Sample $\mathbf{h}$ from the conditional posterior given the observed data
- Sample *fantasy data* $\hat{\mathbf{v}}$ from the conditional posterior given $\mathbf{h}$
- Compute the expectation $\mathbb{E}_{p(\hat{\mathbf{h}}|\hat{\mathbf{v}})}[\hat{\mathbf{h}}]$ for the fantasy data
- Use $\hat{v}_r \mathbb{E}[\hat{h}_k|\hat{\mathbf{v}}]$ to approximate the latter term of the gradient
- (Could also repeat the process a few times and average)

The gradient is hence $\mathbf{v}\mathbb{E}[\mathbf{h}|\mathbf{v}]^T - \hat{\mathbf{v}}\mathbb{E}[\hat{\mathbf{h}}|\hat{\mathbf{v}}]^T$ – we can see it is small if the network can reconstruct the data well by producing fantasy data that looks like the real one

We can also use continuous variables for $\mathbf{v}$ and/or $\mathbf{h}$ – see Section 27.7

More layers?

Two different deep learning models using RBMs as the basic building block

Deep Boltzmann machine is the direct generalization with probability density

$$p(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3)}$$

Deep belief network stacks several RBMs on top of each other and trains them greedily layer-by-layer, only specifying the pairwise potentials

The latter is not multilayer RBM in itself

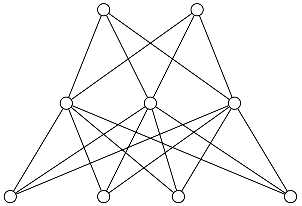
Deep Boltzmann machines

The energy function is given simply by

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = -\mathbf{v}^T \mathbf{W}^1 \mathbf{h}^1 - (\mathbf{h}^1)^T \mathbf{W}^2 \mathbf{h}^2 - (\mathbf{h}^2)^T \mathbf{W}^3 \mathbf{h}^3$$

The conditional densities of the odd layers given the even layers are simple, and vise versa:

$$p(\mathbf{h}^1_i | \mathbf{v}, \mathbf{h}^2) = \mathrm{Ber}(\mathrm{sigm}(\mathbf{v}^T \mathbf{W}^1_{:,i} + \mathbf{W}^2_{i,:} \mathbf{h}^2))$$



Can be pre-trained layer-by-layer

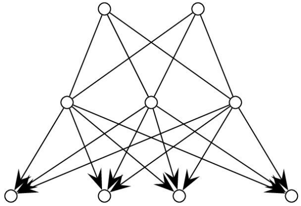
Deep belief networks

Deep belief network combines both directed and undirected edges in a deep network; they mostly started the deep learning hype but are not very widely used today

First two layers with undirected egdes, the rest of the layers with directed edges pointing towards the observed data

Specifies the probability density

$$p(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) = \prod_i \mathrm{Ber}(\mathbf{v}_i | \mathrm{sigm}(\mathbf{h}_1^T \mathbf{W}_{1i})) \prod_j \mathrm{Ber}(\mathbf{h}_{1j} | \mathrm{sigm}(\mathbf{h}_2^T \mathbf{W}_{2j})) \frac{1}{Z} e^{\sum_{kl} \mathbf{h}_{2k}\mathbf{h}_{3l}\mathbf{W}_{3kl}}$$



Deep belief networks

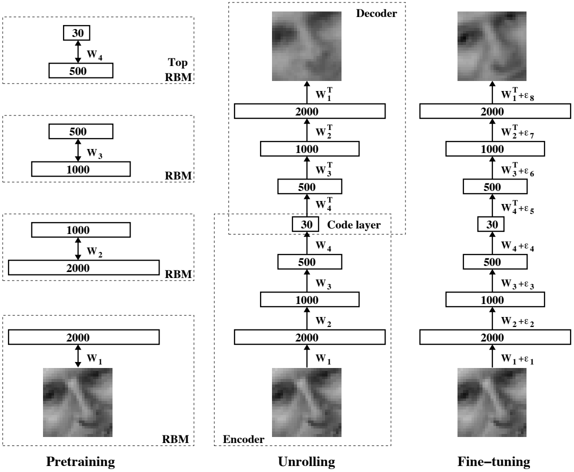
We can train the network layer-by-layer:

- Consider first the observed layer and the first hidden layer; train RBM for this
- Fix the weights for the first layer and compute $\mathbb{E}[h|v]$
- Train the weights between the first and second hidden layer using the above expectations as "input"
- Add more layers until you are happy with the depth

IN the end, we can fine-tune the whole network using gradients (but still need some Gibbs sampling for the top layers)

Deep autoencoder

Deep belief network becomes an autoencoder if we use W^T as weights for a reverse version of the network, possibly fine-tuning the whole network with gradients in the end



Supervised representation learning

Supervised deep networks often learn interesting internal representations for the hidden layers

These are called *distributed representations* (compare to symbolic representation, or cluster-based representations)

We can use those as “unsupervised” feature extraction for other tasks; they are probably good features if they helped in solving the supervised task

If you need vectorial representations for images – for any purpose – you can use the last hidden layer(s) of a CNN trained to classify images

Even better representations obtained if we use richer supervision; deep learning is especially suitable for multi-task learning

Supervised representation learning

We can even start with the goal of learning the representation and use some artificial task for training the network

For example, we can learn *word embeddings* to represent natural language words as vectors by predicting the right context from the left context, or by predicting whether a sequence of words forms a valid sentence

Word embeddings learnt by suitable models allow arithmetics on words: W("woman") - W("man") + W("aunt") = W("uncle")

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza