SoSe 2024

Machine Learning 2/2-X

Technische Universität Berlin

Lecture 5

Representation Learning

Representation Learning

- **Goal:** Representing data in a way that is more efficient for some purpose, for example,
  - Compression, summarization
  - Correlating data to another data modality (e.g. text)
  - Identifying independent components (disentangling the data)
  - Extracting high-level (semantically related) features from the data

Methods for Representation Learning

Methods presented so far in ML1 and ML2:

- Principal component analysis (PCA)
- Canonical correlation analysis (CCA)
- Independent component analysis (ICA)

Today's lecture:

- Sparse coding
- Autoencoders
- Advanced models (normalizing flows, U-Net, etc.)

Sparse Coding

Sparse coding aims to represent data (e.g. an image) as a vector with many elements that are zero. The data can be reconstructed from the sparse code using a dictionary W.

Sparse coding has several advantages:

- **Low storage cost:** A vector with many zeros can be repesented compactly. E.g. the vector given in dense form as $\boldsymbol{s} = (12, 0, 7, 0, 0)$ can be written more compactly as a collection of index-value pairs, i.e. $\boldsymbol{s} = \{0 \to 12; 2 \to 7\}$.
- **Interpretable:** Empirically, the representation learned by sparse coding often disentangles well the different factors of variation in the data, e.g. sources correspond to spatially well-defined visual patterns.

Sparse Coding

Linear sparse coding

- Assume $\boldsymbol{x} \in \mathbb{R}^d$, and its correspond source code is a sparse vector $\boldsymbol{s} \in \mathbb{R}^h$.
- Using the dictionary W (a matrix of size $d \times h$), linear sparse coding reconstructs approximately the data from the source code as:

$$\begin{aligned} \hat{\boldsymbol{x}} &= W\boldsymbol{s} && \text{(dense formulation)} \\ &= \sum_{i \mid s_i \neq 0} W_{:,i}\, s_i && \text{(sparse formulation)} \end{aligned}$$

- *Question:* how do we *learn* the dictionary W and the sparse code $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N$ associated to some dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Sparse Coding Objectives

Linear sparse coding (L_0 formulation)

- Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$ be a dataset and $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^h$ be the representations (sources) of each data point. Let $W \in \mathbb{R}^{d \times h}$ be a dictionary that reconstructs the data from the sources.
- An objective function that implements the sparse coding idea is given by:

$$\min_{W, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_N} \frac{1}{N} \sum_{i=1}^{N} \Big[\underbrace{\|\boldsymbol{x}_i - W\boldsymbol{s}_i\|^2}_{\text{reconstruction}} + \underbrace{\lambda \|\boldsymbol{s}_i\|_0}_{\text{sparsity}} \Big]$$

where the $||\cdot||_0$ is the 0-"norm" (a generalization of the p-norm with $p = 0$). The 0-"norm" counts the non-zero elements in the vector, and therefore penalizes non-sparse solutions. (*Problem:* $\|\boldsymbol{s}_i\|_0$ is nonconvex and is not differentiable.)

Sparse Coding Objectives

Linear sparse coding (L_1-relaxation)

- *Idea:* Replace $\|s_i\|_0$ by $\|s_i\|_1$:

$$\min_{W, s_1, \ldots, s_N} \frac{1}{N} \sum_{i=1}^{N} \Big[\underbrace{\|x_i - W s_i\|^2}_{\text{reconstruction}} + \underbrace{\lambda \|s_i\|_1}_{\text{sparsity}} \Big]$$

- *Advantage:* Given a dictionary W, finding the sources $s_1, \ldots, s_N$ of all data points is now a convex optimization problem (because $\sum_i \|s_i\|_1$ is convex). The objective is also differentiable almost everywhere.
- *Limitation:* The objective is not the same anymore. $\Rightarrow$ Using $\|s_i\|_1$ will produce a sparse solution but not a maximally sparse solution.
- *Another problem:* The second term can be made arbitarily small by scaling up W and scaling down s_i accordingly, i.e. the sparsity penalty is not effective.

Sparse Coding Objectives

Linear sparse coding (L_1 relaxation)

- *Idea:* Add a regularization term on the weights.

$$\min_{W, s_1, \ldots, s_N} \frac{1}{N} \sum_{i=1}^{N} \Big[\underbrace{\left\| x_i - W s_i \right\|^2}_{\text{reconstruction}} + \underbrace{\lambda \| s_i \|_1}_{\text{sparsity}} \Big] + \eta \| W \|_F^2$$

- This resolves the scaling problem between s_i and W, by penalizing large Ws.

Toy example

Consider some dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^2$ and we perform source coding with sources $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^3$. After optimization, most data points are sparse in source space (aligned with the coordinate system), but still reconstruct reasonably well the data.

Toy example

Effect of the sparsity parameter λ on the reconstructed data:

- A low parameter λ perfectly reconstructs the data but does not produce a sparse representation.
- A high parameter λ produces very sparse representation but does not reconstruct the data.
- A intermediate value of λ produces the desired tradeoff.

Convolutional Sparse Coding

Idea: Code images as the convolution of multiple sparse feature maps. Dictionary W is now a set of convolutional filters.

The same convolution filter can be used for any location in pixel space ($\rightarrow$ statistically efficient).

Convolutional Sparse Coding

Standard Sparse Coding:

$\boldsymbol{s}_i$ is a vector, W is a matrix, and $\boldsymbol{x}_i = W \cdot \boldsymbol{s}_i$.

Convolutional Sparse Coding:

$\boldsymbol{s}_i$ is a collection of images, W is a collection of convolution filters, and $\boldsymbol{x}_i = \sum_{j=1}^{h} W_j * \boldsymbol{s}_{ij}$

(Kavukcuoglu'10, Learning Convolutional Feature Hierarchies for Visual Recognition)

Observation: Standard sparse coding learns many replicas of the same filter at different locations. Convolutional coding doesn't have to do this, and instead learns a richer set of features.

Topological Sparse Coding

Additional constraint: Sources dimensions $\boldsymbol{s}_{:,1}, \ldots, \boldsymbol{s}_{:,h}$ must correlate on some predefined 2D grid (topology).

Image source: Yann LeCun, 2012, Learning Invariant Feature Hierarchies

Implementing Sparse Coding

Observation: The total number of variables to optimize is $N \times h$ for the sources and $d \times h$ for the dictionary. For large N, this may not fit in memory or lead to slow optimization. Therefore, two different approaches can be considered depending on whether N is small or large.

Batch algorithm
(works well for small N).

```
Initialize
W, s_1, ..., s_N
while not converged
do
  update s_1, ..., s_N
  update W
end while
```

Online algorithm
(works well for N large).

```
Initialize W
while W not converged do
  Choose i randomly
  infer s_i
  update W approximately from
  s_i
end while
```

Many possible implementations of sparse coding, including efficient ones (cf. Lee et al. 2006, Efficient sparse coding algorithms).

Auto-Encoders

Idea: Facilitate training with an encoding function

$$\boldsymbol{s}_i = V\boldsymbol{x}_i$$

and learn this encoding function from the data.

Objective becomes:

$$\min_{W,V} \quad \frac{1}{N}\sum_{i=1}^{N}\Big[\underbrace{\|\boldsymbol{x}_i - WV\boldsymbol{x}_i\|^2}_{\text{reconstruction}} + \underbrace{\lambda\|V\boldsymbol{x}_i\|_1}_{\text{sparsity}}\Big] + \eta\|W\|_F^2$$

It only depends on the matrices W and V. These matrices can be learned via (stochastic) gradient descent.

Problem: The linear encoding function may not have sufficient representation power to produce sparse representations.

Auto-Encoders with Sparsifying NonLinearity

Idea: Use a *nonlinear* encoding function

$$\boldsymbol{s}_i = \max(0, V\boldsymbol{x}_i)$$

where $\max(0, \cdot)$ is applied element-wise. This function produce values exactly zero for every negative input ($\rightarrow$ helps sparsity!).

The objective becomes:

$$\min_{W,V} \frac{1}{N} \sum_{i=1}^{N} \Big[\underbrace{\left\| \boldsymbol{x}_i - W \max(0, V\boldsymbol{x}_i) \right\|^2}_{\text{reconstruction}} + \underbrace{\lambda \|\boldsymbol{s}_i\|_1}_{\text{sparsity}} \Big] + \eta \|W\|_F^2$$

To avoid creating dead units, we can add an 'entropy term' that forces each unit to activate at least a few times.

Auto-Encoders with Multiple Layers

For complex dataset, components can be better represented using more complex encoding and decoding functions, e.g. each of them consisting of multiple layers.

$$\min_{f,g} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - f \circ g(\boldsymbol{x}_i) \right\|^2$$

- The linear projections are replaced by general functions f and g (typically multilayer network).
- This allows to learn better codes compared to shallow autoencoders.
- A deep autoencoder can be trained using error backpropagation.

Adding Probabilities with Normalizing Flows

Reference publication:
Rezende et al. (2015)
Variational inference with normalizing flows, ICML

Image adapted from: `https://lilianweng.github.io/posts/2018-10-13-flow-models/`

Idea: Structure the encoder in a way that the decoder and encoder are inverse functions.

Change-of-variables theorem (adapted, simplified):

Assume $\pi(\boldsymbol{z})$ a known probability density function on $\boldsymbol{z}$. Assume an invertible decoding function $\boldsymbol{x} = f(\boldsymbol{z})$. The input density function of $p(\boldsymbol{x})$ is given by:

$$p(\boldsymbol{x}) = \pi(\boldsymbol{z}) \cdot \left| \det \frac{d\boldsymbol{z}}{d\boldsymbol{x}} \right| = \pi(f^{-1}(\boldsymbol{x})) \cdot \left| \det \frac{df^{-1}}{d\boldsymbol{x}} \right|$$

Observation: Data distributions in input and source space can be related!

Adding Probabilities with Normalizing Flows

Image adapted from: https://lilianweng.github.io/posts/2018-10-13-flow-models/

By iterative application of the change-of-variables theorem, we get:

$$\log \underbrace{\pi_K(\boldsymbol{z}_K)}_{p(\boldsymbol{x})} = \log \pi_0(\boldsymbol{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{\partial f_i}{\partial z_{i-1}} \right|$$

Derivations can be found at:

https://lilianweng.github.io/posts/2018-10-13-flow-models/.

Adding Probabilities with Normalizing Flows

Given some dataset $\mathcal{D}$, the model can be trained for *maximum likelihood*, i.e.

$$\begin{aligned}
&\arg\max_f \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \log p(\boldsymbol{x}) \\
&\quad = \arg\max_f \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \left[\log \pi_0(f^{-1}(\boldsymbol{x})) - \sum_{i=1}^{K} \log \left| \det \frac{\partial f_i}{\partial z_{i-1}} \right| \right]
\end{aligned}$$

where π_0 is set to some base distribution, e.g. multivariate Gaussian. *(Note: the functions f_i should be easily invertible, and the determinant of their Jacobian easy to compute.)*

Takeaways:

- Not only the available data can be encoded and reconstructed (as for standard autoencoders),
- ... but one can now sample from the source distribution in a way that reflects the true distribution $p(\boldsymbol{x})$. (i.e. can be used as a generative model).

Learning Invariant Representations

- In practice, sparsely encoding and accurately reconstructing/generating the data may not be the end goal.
- We may instead want to learn a representation that satisfies additional properties, e.g. being invariant to factors of variations that are irrelevant for the task (e.g. small translations or rotations).

Local Invariance with Denoising Autoencoders

The denoising autoencoder can be seen as a standard deep autoencoder to which we have added a first layer that adds noise to the input.

$$\min_{f,g} \frac{1}{N} \sum_{i=1}^{N} \left[\left\| \boldsymbol{x}_i - f \circ g \circ \text{noise}(\boldsymbol{x}_i) \right\|^2 \right]$$

Reference publication: Vincent et al. 2010, Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, JMLR

Local Invariance with Denoising Autoencoders

Training a denoising autoencoder produces a representation $f(\boldsymbol{x})$ that is insensitive to small local variations and therefore more useful for prediction tasks (promotes large margins between data manifolds).

Learning Abstract Representations

- Low-level information (that doesn't require many layers of abstraction) can flow through a shortcut connection.
- Only information that requires more layers (often abstract concepts) are retained in the representation.
- Models that implement this (or similar) idea include the U-Net (Ronneberger 2015), the Deep Boltzmann Machine (Salakhutdinov 2009), and the Ladder Network (Rasmus 2015).

Shortcut Connections for Image Segmentation

U-Net Model: *Shortcut connections* finely encode the boundaries of the segments. *Abstract representation* is more complex and spatially downsampled, and serves to understand the segment's content. *Both* are necessary to achieve good segmentation.

Reference publication: Ronneberger et al. (2015), U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI

Shortcut Connections for Image Segmentation

Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

Representation-Based Transfer Learning

Idea: Instead of reconstructing the data, use an *auxiliary task* for learning the representation. The learned representation is then transferred to the target task.

Note: For transfer learning to work, the auxiliary task should typically be *related* to the target task and also *more general* (e.g. more difficult or with more classes).

Representation-Based Transfer Learning

Idea: Generate an *artificial task* for learning the representation. This is known as *self-supervised learning*. The task should be complex enough so that solving it produces useful features.

Example of auxiliary tasks for image classification:

- Learn to inpaint or colorize an image.
- Learn to predict whether two images are related or not.
- Learn to map images into dense and well-separated clusters.

No labels needed! Only creativity in setting up the self-supervised tasks.

Learning Domain Invariant Representations

In practice, data often comes from multiple domains. We want a model that predicts well for each domain (even those that are the most difficult or that have fewer labels).

Learning Domain Invariant Representations

- *Approach:* learn a function F that maps the data to a representation where the domains cannot be differentiated, and from which a domain-invariant classifier C can be built.
- Domain invariance can be measured by how low the *Wasserstein distance* between distributions of to each domain are.
- *The Wasserstein distance $W(\mathcal{P}, \mathcal{Q})$ measures how much it costs (in Euclidean sense) to transport every bit of distribution $\mathcal{P}$ onto $\mathcal{Q}$.*

Reference publication: Andéol et al. (2021) Learning Domain Invariant Representations by Joint Wasserstein Distance Minimization

Learning Domain Invariant Representations

Insight: The Wasserstein distance (on the joint representation-label space) between the two domains $\mathcal{P}$ and $\mathcal{Q}$, i.e.

$$W(\mathcal{P}_{\mathcal{Z}\mathcal{Y}}, \mathcal{Q}_{\mathcal{Z}\mathcal{Y}})$$

can be upper-bounded in a way that it becomes expressible as combination of common and easily optimizeable neural network modules (classifier, domain critic, etc.):

In this architecture, domain invariance is achieved by simultaneously (1) learning to detect the domain from the representation (domain critic D), and (2) learning a feature extractor F that *fools* the domain critic D.

Learning Domain Invariant Representations

The process of learning a domain invariant representation can be visualized by applying UMAP (a low-dimensional embedding method similar to T-SNE) in the representation space, and color-coding points by their domain.

Before training, the representation is strongly dominated by domain membership (MNIST: black&white handwritten digits, SVHN: color printed digits). After training, distributions are becoming aligned (hard to distinguish between domains) and form clusters corresponding to the different classes.

Representation-Based Transformations

Suppose we would like our network to predict not the input image itself, but the transformation between two images.

Can we represent the way two images are deformed rather than the content of the image itself. I.e. can we learn a representation of transformation that is invariant to the actual image content?

Representation-Based Translation

Translation can be detected using a convolution:

$$\mathcal{T}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} * \boldsymbol{y}$$

The same operation can also be computed as a product using the Fourier theorem:

$$\mathcal{T}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{x}) \times \mathcal{F}(\boldsymbol{y}))$$

Idea: build a network with these product structures, and learn $\mathcal{F}$ from the data.

Representation-Based Translation

The autoencoder consists of two parts (1) an encoder that predict the shift, (2) a decoder that applies the shift to translate the input $\boldsymbol{x}$. For each operation, we make use of a product operation.

The autoencoder can be trained end-to-end, e.g. by minimizing the mean-square error between the predicted and true translation.

Representation-Based Translation

Source: Memisevic'10, Learning to Represent Spatial Transformations

Observation: Filters learned by the encoder (left) and decoder (right) look like 2D Fourier decompositions, where the two branches are similar but with small spatial offsets.

Representation-Based Rotation

Different filters can be obtained if the model is trained on different data transformations. For example, when training the model with pairs of *rotated* images as input, we get *spiral*-like filters in the encoder and decoder:

Source: Memisevic'10, Learning to Represent Spatial Transformations

The approach can be generalized to represent any transformation (e.g. deformations, motion in video, etc.).

Summary

- Similar to PCA, CCA, and ICA, sparse coding and autoencoders learn representations from *unsupervised* data.
- Autoencoders can be trained quickly and can be implemented as neural networks. Thus, they can make use of all neural network layers available such as convolution, pooling, etc.
- Autoencoders can serve different purposes, e.g. produce compact (sparse) representations, generating new data, learning invariant representations, learning data transformations.