Statistical Machine Learning

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2015/2016 // Lecture 11

Representation Learning

Task: *nearest-neighbor classification*, or *information retrieval*

What distance to use? Euclidean might not be the best...

Task: *nearest-neighbor classification*, or *information retrieval*

What distance to use? Euclidean might not be the best...

Metric Learning

Given data $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$, find a **distance function**, $d(x, \bar{x}) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that

$$d(x_i, x_j) \text{ small} \quad \Leftrightarrow \quad y_i = y_j$$

Special case: Mahalanobis Metric Learning

For $\mathcal{X} \subset \mathbb{R}^d$, parameterize

$$d_M^2(x, \bar{x}) = (x - \bar{x})^\top M (x - \bar{x})$$

and learn positive (semi-)definite $M \in \mathbb{R}^{d \times d}$.

Relevant Component Analysis (RCA) [Bar-Hillel *et al.*, 2003]

Given group $X^1 = \{x^1_1, \dots, x^1_{n^1}\}$, ..., $X^K = \{x^K_1, \dots, x^K_{n^K}\}$ (e.g. $X^k = \{x_i : y_i = k\}$ all example of each label). Solve

$$\min_{M \succcurlyeq 0} \quad \sum_{k=1}^{K} \sum_{i=1}^{n^k} d^2_M(x^k_i, m^k) \quad \text{subject to} \quad \det M \geq 1.$$

with $m^k = \frac{1}{n^k} \sum_{i=1}^{n^k} x^k_i$.

- pull examples of same class together
- but prevent overall volume from shrinking to 0

Disadvantages:

- optimizing over all positive definite matrices is hard
- enforces low *average distance*, but outliers might exist that hurt k-NN performance

Large Margin Nearest Neighbor (LMNN) [Weinberger *et al.*, 2006]

Given data $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$, solve

$$\min_{M \succcurlyeq 0, \xi \geq 0} \quad \sum_{i,j=1}^{n} d_M^2(x_i, x_j) \; + \; \sum_{i,j,l} \xi_{ijl}$$

subject to, for all i, j, l with $y_i = y_j \neq y_l$,

$$d_M^2(x_i, x_l) - d_M^2(x_i, x_j) \geq 1 - \xi_{ijk}$$

- pull examples together,
- but enforce examples of different classes to have larger distance from each other than examples of same class
- convex optimization problem

Disadvantages:

- optimizing over all positive definite matrices is still hard

Distance Learning, Alternative Views

Minimizing a function over the set of all positive definite matrices:

- difficult, since matrix entries must fulfill many constraints of high order (e.g. $\det M \geq 0$)
- but: set is convex, so no danger of getting stuck in local optima

Distance Learning, Alternative Views

Minimizing a function over the set of all positive definite matrices:

- difficult, since matrix entries must fulfill many constraints of high order (e.g. $\det M \geq 0$)
- but: set is convex, so no danger of getting stuck in local optima

Alternative: parameterize $M = L^\top L$ with $L \in \mathbb{R}^{m \times n}$

- simpler, since M automatically positive definite for any L
- enforcing a low rank on M is easy, since rank$(M) \leq m$
- but: non-convex objective, only local optimum might be found

Minimizing a function over the set of all positive definite matrices:

- difficult, since matrix entries must fulfill many constraints of high order (e.g. $\det M \geq 0$)
- but: set is convex, so no danger of getting stuck in local optima

Alternative: parameterize $M = L^\top L$ with $L \in \mathbb{R}^{m \times n}$

- simpler, since M automatically positive definite for any L
- enforcing a low rank on M is easy, since rank$(M) \leq m$
- but: non-convex objective, only local optimum might be found

Other interpretation: **learn a representation**, $x \mapsto Lx$, because

$$
\begin{aligned}
d^2_M(x, \bar{x}) &= (x - \bar{x})^\top M (x - \bar{x}) = (x - \bar{x})^\top L^\top L (x - \bar{x}) \\
&= (Lx - L\bar{x})^\top (Lx - L\bar{x}) = d^2_{Eucl}(Lx, L\bar{x})
\end{aligned}
$$

Common problem in *signal processing*:

Coding

Let $D = \{d_1, \dots, d_m\} \subset \mathbb{R}^d$ be a *dictionary*, e.g. of *typical signals*.
Given $x \in \mathbb{R}^d$, find coefficients $\alpha_1, \dots \alpha_m$, such that

$$x \approx \sum_{j=1}^{m} \alpha_j d_j$$

Typical Cases:

- no constraints on α:

$$\mathbf{min}_\alpha \|x - \sum_{j=1}^{m} \alpha_j d_j\|^2$$

$\rightarrow$ linear algebra, project x to span$(d_1, \dots, d_m)$

Typical Cases:

- enforce *sparsity* in α

$$\min_{\alpha} \sum_{j=1}^{m} |\alpha_j| \qquad \text{subject to} \quad \|x - \sum_{j=1}^{m} \alpha_j d_j\| \leq \epsilon$$

  or

$$\min_{\alpha} \|x - \sum_{j=1}^{m} \alpha_j d_j\|^2 + \lambda \sum_{j=1}^{m} |\alpha_j|$$

  called **Sparse Coding**,

- popular, e.g., in Neuroscience
  - α_j are neuron firing rates,
  - *sparsity* expresses that at every time only a few neurons fire

Representation Learning: Sparse Coding

What, if we don't know $D = \{d_1, \ldots, d_m\} \subset \mathbb{R}^d$? Learn it from data!

Dictionary Learning

Given $x_1, \ldots, x_n$:

$$\min_{\alpha, D} \sum_{i=1}^{n} \|x_i - \sum_{j=1}^{m} \alpha_j^i d_j\|^2 + \lambda \sum_{j=1}^{m} |\alpha_j|$$

Solve by *alternating optimization*

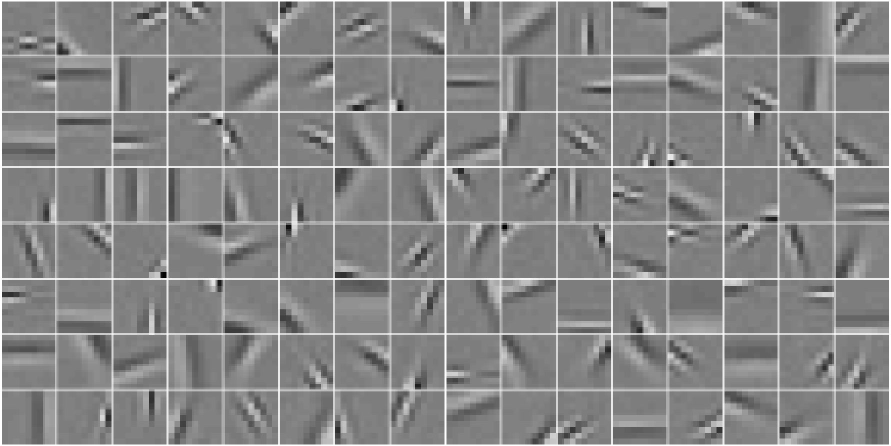
- initialize D (e.g. random elements x_i)
- repeat
  - solve for α with fixed D
  - solve for D with fixed α
- until convergence

Convergences to local optimum, multiple restarts for better results

Example: Dictionary Learning

For $x_1, \ldots, x_n$ **image patches**.

Learned dictionary:

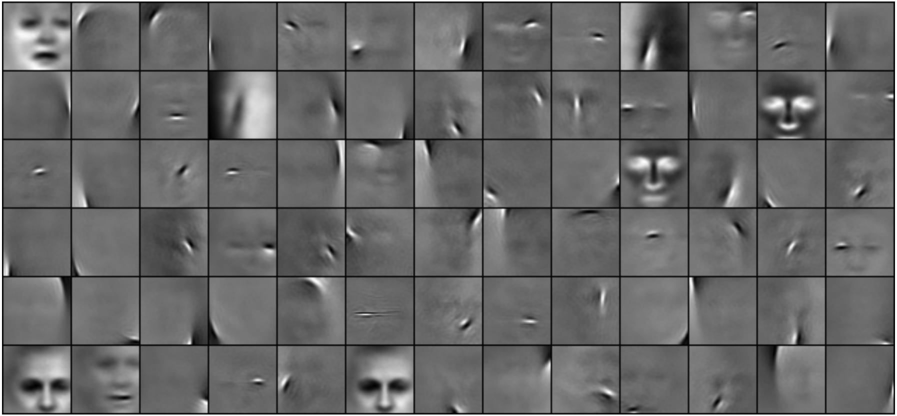


(claim: human visual system has similar representation)

Example: Dictionary Learning

For $x_1, \ldots, x_n$ **images of faces**.

Learned dictionary:



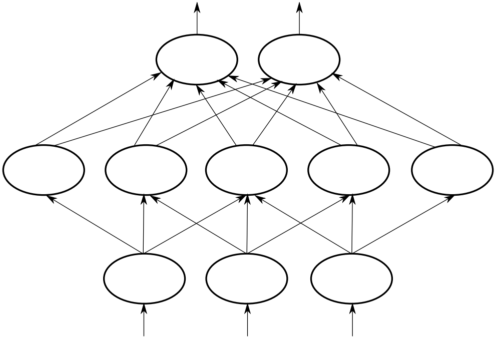
(used, e.g., in face recognition systems)

Deep Learning

Artificial Neural Network have been proposed as promising models to achieve artificial intelligence since the 1950s.

Main idea:

- stack layers of simple elements ("neurons")
- (part of) one layer's output is the next layer's input.



Models differ in:

- network topology (number of layers and neurons, connectivity)
- neuron output (binary or real-valued)
- parameterization of each neuron

Main step: end-to-end training across L layers

$$f(x) = h_L \circ g_L \cdots \circ h_1 \circ g_1(x)$$

where

- each $g_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ is linear, i.e. $g_l(x) = W_l x$ for $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$
- each $h_l : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$ is non-linear but acts componentwise, e.g.

$$h_l(z) = (\,\sigma(z[1]), \dots, \sigma(z[l])\,)$$

for $\sigma(t) = \tanh(t)$ or $\sigma(t) = \mathbf{max}\{0, t\}$ or $\sigma(t) = \frac{1}{1+e^{-t}}$

Trained end-to-end by minimizing a loss function:

$$\min_{W_1,\dots,W_L} \quad \sum_{i=1}^{n} \ell(\,y_i, f(x_i)\,)$$

- typically: stochastic gradient descent (with mini-batches).
- organize computations efficiently: **backpropagation** algorithmic

Example 1: Deep Autoencoder Networks [Hinton et al, 2006]

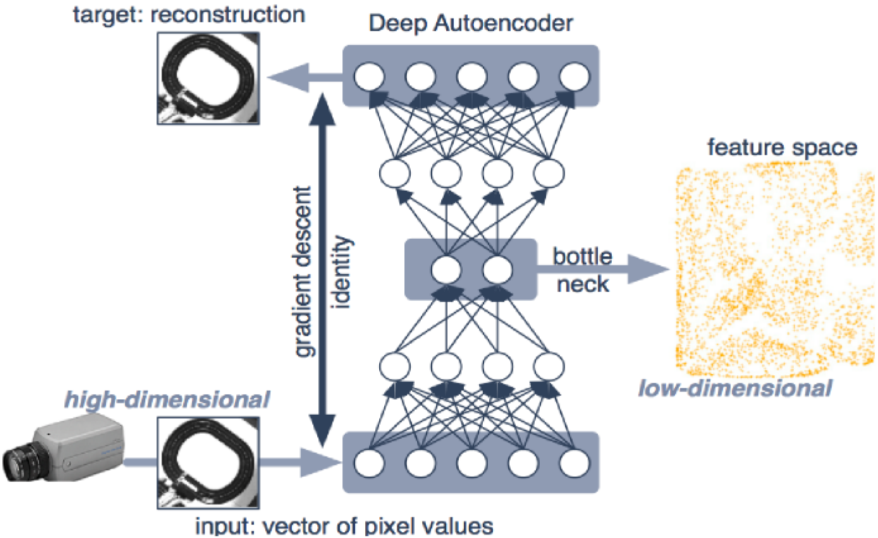


Image: http://wiki.ldv.ei.tum.de

Example 1: Deep Autoencoder Networks [Hinton et al, 2006]

Given: data $\{x^1, \dots, x^n\} \subset \mathbb{R}^d$
Goal: learn a new data representation, $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$

- symmetric topology, $(2K + 1)$ layers,
  layer k and $2K + 2 - k$ are mirrored copies of each other
- layers are fully connected to each other
- number of neuron per layer decreases from outer to inner layers
- each neuron i is a *stochastic* binary function, f_i, of its input x:

$$p(f_i = 1|x) = \frac{1}{1 + \exp(-a_i)} \qquad \text{sigmoid (logistic function)}$$

$$\text{with} \quad a_i(x) = \sum\nolimits_j w_{ij} x_j + b_j \qquad \text{"activation"}$$

- objective is reconstruction error, $\sum_i \|x^i - F(x^i)\|^2$ (non-convex)
  - first, train each layer as separate auto-encoder
  - then, train jointly by gradient descent
  - also possible: binary-valued outputs ("restricted Boltzman machine")
- after training, *middle layer* is new data representation

Demo:

http://dpkingma.com/sgvb_mnist_demo/demo.html

Example 2: Convolutional Neural Networks (CNNs) [LeCun et al, 1980s]

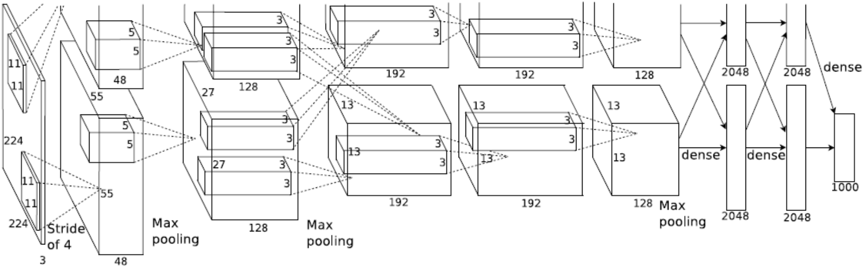
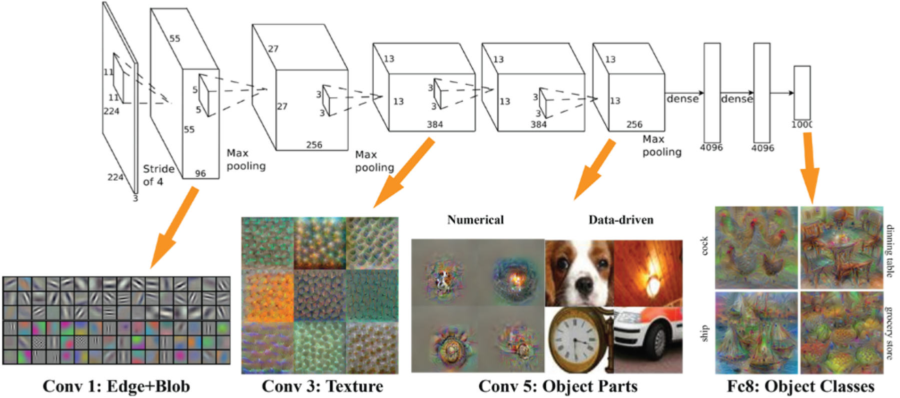


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

Example 2: Convolutional Neural Networks (CNNs) [LeCun et al, 1980s]



Example 2: Convolutional Neural Networks (CNNs) [LeCun et al, 1980s]

Given: training set $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathbb{R}^d \times \mathcal{Y}$
Goal: learn a classifier, $G : \mathbb{R}^d \to \mathcal{Y}$

- each neuron i is a *real-valued* function, f_i, of its input x:

$$F_i(x) = \mathbf{max}\{0, a_i\} \qquad \text{Rectified Linear Unit (ReLU)}$$

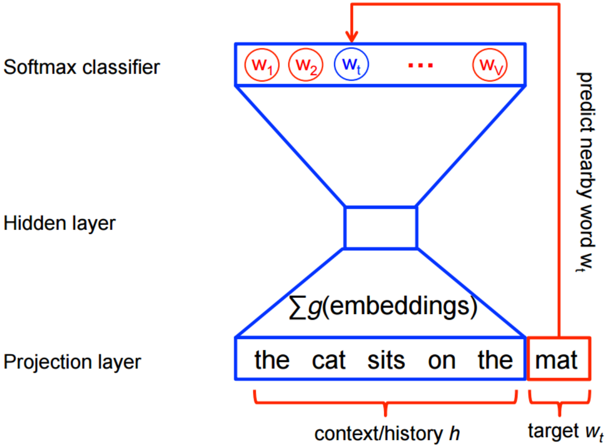
$$\text{with} \quad a_i(x) = \sum_j w_{ij} x_j + b_j \qquad \text{"activation"}$$

- first layers **convolutions**,
  - shared weights w_{ij}, acting on different subwindows of the layer's input
- last few layers **fully connected**
  - individual weights for every neuron-neuron connection
- last layer: one neuron output, F_k, per target class
- objective is squared or log-loss
  - non-convex optimization w.r.t. most parameters
  - train jointly by gradient descent, often with GPU support
  - additional tricks: weight decay, momentum term, dropout, batch normalization,...
- after training, $G(x) = \mathbf{argmax}_y F_y(x)$

Demo:

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Example 3: word2vec [Mikolov et al, ICLR 2013]



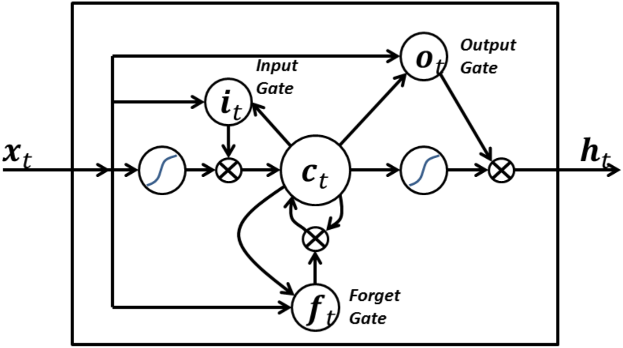
For every (e.g. English) word, learn a vector $w_i \in \mathbb{R}^d$, such that it is "easy" to predict the next word of a text from a short history.

Image: https://www.tensorflow.org/versions/r0.8/tutorials/word2vec/index.html

Demo:

http://deeplearner.fz-qqq.net/

Example 4: Long-Short Term Memory Networks (LSTM) [Hochreiter, Schmidthuber, 1997]



Each "neuron" has a memory cell that can keep its value indefinitely.
Access controlled by additional inputs: `store` or `erase` a value.

- network handles sequential inputs/outputs with 'long-term' memory
- internal weights can be used as representation for a sequence

Demo:

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Summary – Representation Learning

Representation Learning is a recent trend in Machine Learning:

- **Metric learning** is well understood
  - for linear (Mahalanobis), convex formulations exist
  - can vastly improve quality, e.g., of nearest neigbhor classifiers,
  - less impact on linear classifiers that learn per-coordinate weights, e.g. linear SVMs
- **Dictionary learning** is popular in some application areas, e.g.
  - face recognition (explains one face as mixture of others)
  - computational neuroscience (dictionary elements $\equiv$ neurons), etc.
- recent trend: **Deep Network**
  - learn data representation and classifiers jointly, end-to-end
  - very impressive results in Computer Vision, Speech, Language, ...
  - results have become more reproducible in the last few years,
  - but, still a lot of engineering required to get good results