Lecture 11. Kohonen maps, autoencoders, transfer learning, generative adversarial networks

Alex Avdyushenko

Kazakh-British Technical University

November 19, 2022

Five-minutes block

Five-minutes block

- What are the main disadvantages of the encoder-decoder architecture?
- Write down the attention model formula $Attn(q, K, V)$
- Describe two criteria for BERT training

Formulation of the clustering problem

Given:
$X^\ell = \{x_1, \ldots, x_\ell\}$ — training set of objects $x_i \in \mathbb{R}^n$
$\rho : X \times X \to [0, \infty)$ — distance function between objects

Formulation of the clustering problem

Given:
$X^\ell = \{x_1, \dots, x_\ell\}$ — training set of objects $x_i \in \mathbb{R}^n$
$\rho : X \times X \to [0, \infty)$ — distance function between objects
Find:
Y is the set of clusters, for example, given by their centers $w_y \in \mathbb{R}^n$
Let the clustering algorithm is Winner Takes All (WTA):

$$a(x) = \arg\min_{y \in Y} \rho(x, w_y)$$

Formulation of the clustering problem

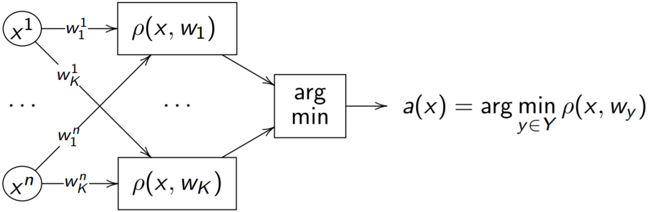
Given:
$X^\ell = \{x_1, \dots, x_\ell\}$ — training set of objects $x_i \in \mathbb{R}^n$
$\rho : X \times X \to [0, \infty)$ — distance function between objects
Find:
Y is the set of clusters, for example, given by their centers $w_y \in \mathbb{R}^n$
Let the clustering algorithm is Winner Takes All (WTA):

$$a(x) = \arg\min_{y \in Y} \rho(x, w_y)$$

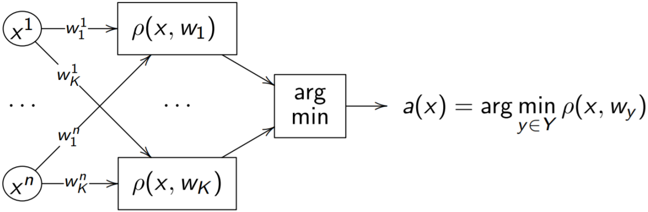
Optimization criterion: average intracluster distance

$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \rho^2(x_i, w_{a(x_i)}) \to \min_{w_y : y \in Y}$$

Kohonen Network — Two Layer Neural Network



Kohonen Network — Two Layer Neural Network



Gradient step in stochastic gradient descent:

$$w_y = w_y + \eta(x_i - w_y)[a(x_i) = y]$$

If x_i belongs to the cluster y, then w_y is shifted towards x_i

T. Kohonen. Self-organized formation of topologically correct feature maps. 1982.

Stochastic Gradient Descent

Input: sample X^ℓ, learning rate η, parameter λ
Output: cluster centers $w_1, \dots, w_K \in \mathbb{R}^n$

Stochastic Gradient Descent

Input: sample X^ℓ, learning rate η, parameter λ
Output: cluster centers $w_1, \dots, w_K \in \mathbb{R}^n$

1. initialize centers $w_y,\ y \in Y$

Stochastic Gradient Descent

Input: sample X^ℓ, learning rate η, parameter λ
Output: cluster centers $w_1, \dots, w_K \in \mathbb{R}^n$

1. initialize centers $w_y,\ y \in Y$
2. functional estimate:

$$Q = \sum_{i=1}^{\ell} \rho^2(x_i, w_{a(x_i)})$$

Stochastic Gradient Descent

Input: sample X^ℓ, learning rate η, parameter λ
Output: cluster centers $w_1, \dots, w_K \in \mathbb{R}^n$

1. initialize centers $w_y,\ y \in Y$
2. functional estimate:

$$Q = \sum_{i=1}^{\ell} \rho^2(x_i, w_{a(x_i)})$$

3. **repeat**
   - select object x_i from X^ℓ (e.g. random)
   - compute cluster: $y = \arg\min_{y \in Y} \rho(x_i, w_y)$
   - make a gradient step: $w_y = w_y + \eta(x_i - w_y)$
   - evaluate functional: $Q = \lambda\rho^2(x_i, w_y) + (1 - \lambda)Q$

Stochastic Gradient Descent

Input: sample X^ℓ, learning rate η, parameter λ
Output: cluster centers $w_1, \dots, w_K \in \mathbb{R}^n$

1. initialize centers $w_y,\ y \in Y$
2. functional estimate:

$$Q = \sum_{i=1}^{\ell} \rho^2(x_i, w_{a(x_i)})$$

3. **repeat**
   - select object x_i from X^ℓ (e.g. random)
   - compute cluster: $y = \arg\min_{y \in Y} \rho(x_i, w_y)$
   - make a gradient step: $w_y = w_y + \eta(x_i - w_y)$
   - evaluate functional: $Q = \lambda\rho^2(x_i, w_y) + (1 - \lambda)Q$
4. **while** the value of Q and/or the weights of w_y do not converge

Hard and soft competition

WTA, Winner Takes All

$$w_y = w_y + \eta(x_i - w_y){\color{red}[a(x_i) = y]}, y \in Y$$

WTA rule disadvantages

- slow convergence rate
- some cluster centers may never be selected

Hard and soft competition

WTA, Winner Takes All

$$w_y = w_y + \eta(x_i - w_y)[a(x_i) = y], y \in Y$$

WTA rule disadvantages

- slow convergence rate
- some cluster centers may never be selected

Soft competition rule (WTM, Winner Takes Most)

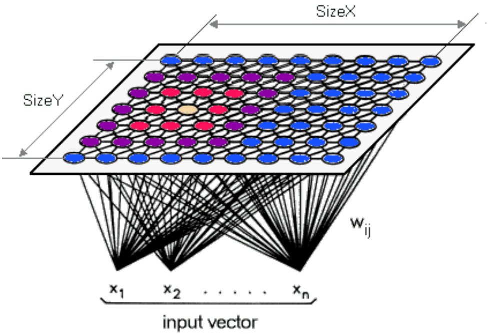
$$w_y = w_y + \eta(x_i - w_y)K(\rho(x_i, w_y)), y \in Y$$

where the kernel $K(\rho)$ is a nonnegative non-increasing function.
Now the centers of all clusters are shifted towards x_i, but the farther from x_i, the smaller the shift

Kohonen Map (Self Organizing Map, SOM)

We introduce a rectangular grid of clusters
$\{1, \dots, SizeX\} \times \{1, \dots, SizeY\}$
Each node (x, y) is assigned a Kohonen neuron $w_{xy} \in \mathbb{R}^n$
Along with the metric $\rho(x_i, w_{xy})$, a metric on the grid is introduced:

$$r((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$



Kohonen Map Training

Input: sample X^ℓ, learning rate η
Output: $w_{xy} \in \mathbb{R}^n$ are weight vectors

Kohonen Map Training

Input: sample X^ℓ, learning rate η
Output: $w_{xy} \in \mathbb{R}^n$ are weight vectors

1. initialize weights: $w_{xy} = \text{random}\left(-\frac{1}{2MH}, \frac{1}{2MH}\right)$

Kohonen Map Training

Input: sample X^ℓ, learning rate η

Output: $w_{xy} \in \mathbb{R}^n$ are weight vectors

1. initialize weights: $w_{xy} = \text{random}\left(-\frac{1}{2MH}, \frac{1}{2MH}\right)$
2. **repeat**
   - choose a random object x_i from X^ℓ
   - WTA: compute cluster coordinates:

$$(a_i, b_i) = \arg\min_{(a,b)} \rho(x_i, w_{ab})$$

   - **for all** $(a, b) \in \text{Neighborhood}\,(a_i, b_i)$

     WTM: do gradient descent step:

     $w_{ab} = w_{ab} + \eta(x_i - w_{ab})K(r((a_i, b_i), (a, b)))$

Kohonen Map Training

Input: sample X^ℓ, learning rate η
Output: $w_{xy} \in \mathbb{R}^n$ are weight vectors

1. initialize weights: $w_{xy} = \text{random}\left(-\frac{1}{2MH}, \frac{1}{2MH}\right)$
2. **repeat**
   - choose a random object x_i from X^ℓ
   - WTA: compute cluster coordinates:

     $$(a_i, b_i) = \arg\min_{(a,b)} \rho(x_i, w_{ab})$$

   - **for all** $(a, b) \in \text{Neighborhood}\,(a_i, b_i)$
     WTM: do gradient descent step:
     $w_{ab} = w_{ab} + \eta(x_i - w_{ab})K(r((a_i, b_i), (a, b)))$
3. **yet** clustering does not stabilize

Interpretation of Kohonen Maps

Two types of graphs — color maps $SizeX \times SizeY$

- Node color (a, b) — local density at point (a, b) — average distance to k closest sample points
- One card for each feature: node color (a, b) — value of j-th component of vector w_{ab}

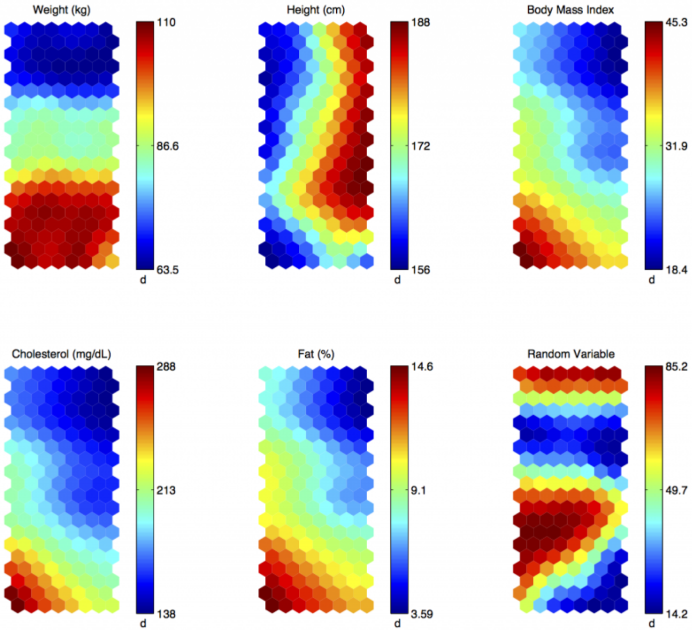
Interpretation of Kohonen Maps

Two types of graphs — color maps $SizeX \times SizeY$

- Node color (a, b) — local density at point (a, b) — average distance to k closest sample points
- One card for each feature: node color (a, b) — value of j-th component of vector w_{ab}

Example

Let's look at Kohonen's maps, built on six features collected from 1000 people.


Weight (kg)
110
86.6
63.5
d
Height (cm)
188
172
156
d
Body Mass Index
45.3
31.9
18.4
d
Cholesterol (mg/dL)
288
213
138
d
Fat (%)
14.6
9.1
3.59
d
Random Variable
85.2
49.7
14.2
d

Pros and cons of Kohonen cards

+ visual analysis of multidimensional data
− **Distortion**. Close objects in the original space can move to far points on the map, and vice versa
− **Subjectivity**. The map depends not only on the cluster data structure, but also on...

- properties of the smoothing kernel
- (random) initialization
- of (random) selection of x_i during iterations

Pros and cons of Kohonen cards

\+ visual analysis of multidimensional data

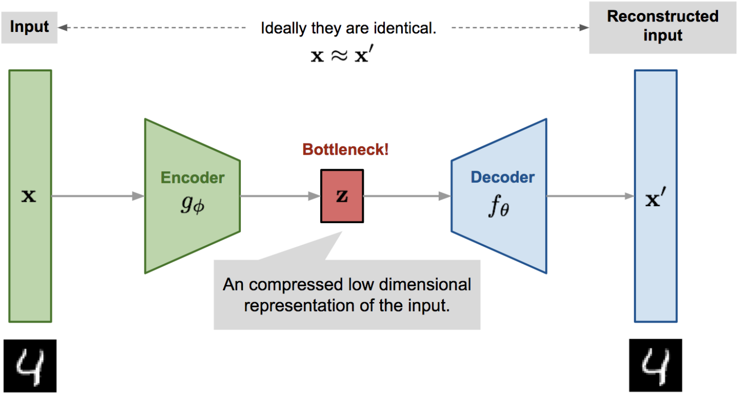
− **Distortion**. Close objects in the original space can move to far points on the map, and vice versa

− **Subjectivity**. The map depends not only on the cluster data structure, but also on...

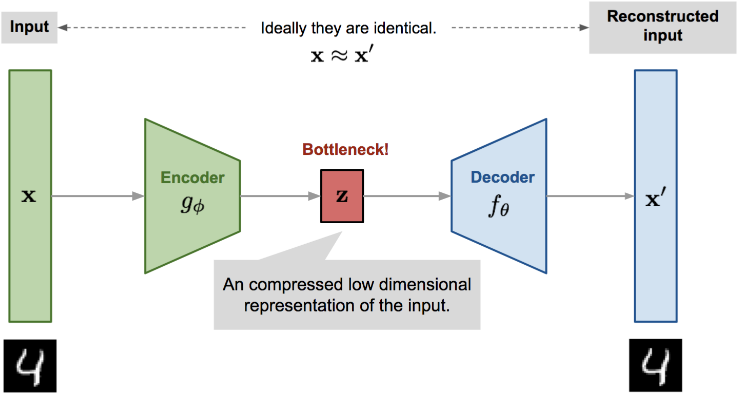
- properties of the smoothing kernel
- (random) initialization
- of (random) selection of x_i during iterations

Well suited for exploratory data analysis (EDA).

Autoencoders



Autoencoders



Question

What can be used as encoder and decoder here?

Ways to use autoencoders

- Feature generation, for example, to effectively solve supervised learning problems
- Dimensionality reduction
- Low loss data compression
- Trainable object vectorization, embeddable in deeper neural network architectures
- Generation of synthetic objects similar to real ones

Rumelhart, Hinton, Williams. Learning Internal Representations by Error Propagation. 1986.

David Charte et al. A practical tutorial on autoencoders for nonlinear feature fusion: taxonomy, models, software and guidelines. 2018.

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

- it is not necessarily $B = A^T$ (although this is often done)

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

- it is not necessarily $B = A^T$ (although this is often done)
- arbitrary A, B instead of orthogonal

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

- it is not necessarily $B = A^T$ (although this is often done)
- arbitrary A, B instead of orthogonal
- non-linear models

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

- it is not necessarily $B = A^T$ (although this is often done)
- arbitrary A, B instead of orthogonal
- non-linear models
- arbitrary loss function $\mathscr{L}$ instead of quadratic

Linear auto encoder and principal component analysis

$$\mathscr{L}_{AE}(A, B) = \sum_{i=1}^{\ell} \|BAx_i - x_i\|^2 \to \min_{A,B}$$

Principal Component Analysis: $F = (x_1 \dots x_\ell)^T$, $U^T U = I_m$, $G = FU$,

$$\|F - GU^T\|^2 = \sum_{i=1}^{\ell} \|UU^T x_i - x_i\|^2 \to \min_{U}$$

The autoencoder generalizes the principal component analysis:

- it is not necessarily $B = A^T$ (although this is often done)
- arbitrary A, B instead of orthogonal
- non-linear models
- arbitrary loss function $\mathscr{L}$ instead of quadratic
- SGD optimization instead of singular value decomposition (SVD)

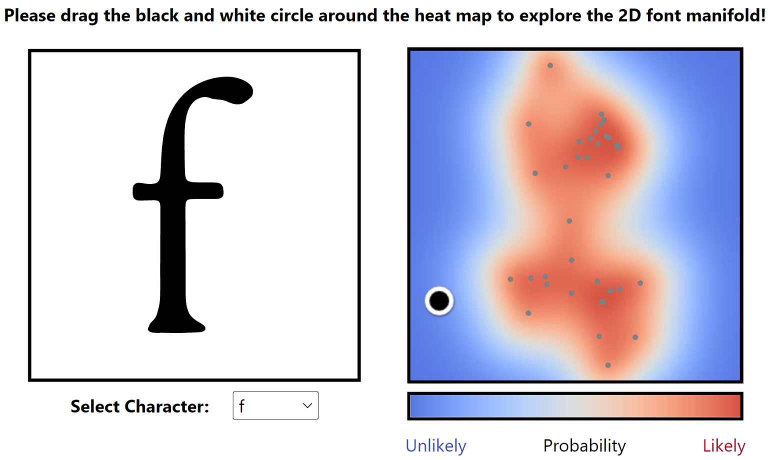
Sparse auto encoder

Reminder from SVM

If the loss function has a kink, then we select objects. If regularizer has a kink, then we select features.

- Applying L_1 or L_2 regularization to weight vectors
- Applying of L_1-regularization to representation vectors $z_i = Ax_i$
- Entropy regularization

D.Arpit et al. Why regularized auto-encoders learn sparse representation? 2015



2d font manifold demonstration

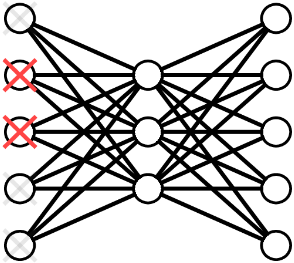
Denoising Auto Encoder

Stability of code vectors z_i with respect to noise in x_i:

$$\mathscr{L}_{DAE}(\alpha, \beta) = \sum_{i=1}^{\ell} E_{\tilde{x} \sim q(\tilde{x}|x_i)} \mathscr{L}(g(f(\tilde{x}, \alpha), \beta), x_i) \to \min_{\alpha, \beta}$$

Instead of calculating the expectation $E_{\tilde{x}}$ in the stochastic gradient method, x_i objects are sampled and noisy one at a time: $\tilde{x} \sim q(\tilde{x}|x_i)$

- Gaussian noise: $\tilde{x} \sim N(x_i, \sigma^2 I)$
- zeroing components of vector x_i with probability p_0



P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. ICML-2008.

Variational Auto Encoder

A generative model is constructed capable of generating new objects x similar to the objects of the sample $X^\ell = \{x_1, \dots, x_\ell\}$
$q_\alpha(z|x)$ — probabilistic encoder with α parameter
$p_\beta(\hat{x}|z)$ — probabilistic decoder with β parameter

$$\mathscr{L}_{VAE}(\alpha, \beta) = \sum_{i=1}^{\ell} \log p(x_i) = \sum_{i=1}^{\ell} \log \int q_\alpha(z|x_i) \frac{p_\beta(x_i|z)p(z)}{q_\alpha(z|x_i)} dz \geq$$

$$\geq \sum_{i=1}^{\ell} \int q_\alpha(z|x_i) \log p_\beta(x_i|z) dz - KL(q_\alpha(z|x_i) \| p(z)) \to \max_{\alpha,\beta}$$

D.P.Kingma, M.Welling. Auto-encoding Variational Bayes. 2013.

C.Doersch. Tutorial on variational autoencoders. 2016.

$$\sum_{i=1}^{\ell} \underbrace{E_{z\sim q_\alpha(z|x_i)} \log p_\beta(x_i|z)}_{\text{quality reconstruction}} - \underbrace{KL(q_\alpha(z|x_i)\|p(z))}_{\text{regularizer by } \alpha} \to \max_{\alpha,\beta}$$

where $p(z)$ is the prior distribution, usually $N(0, \sigma^2 I)$

$$\sum_{i=1}^{\ell} \underbrace{E_{z\sim q_\alpha(z|x_i)} \log p_\beta(x_i|z)}_{\text{quality reconstruction}} - \underbrace{KL(q_\alpha(z|x_i)\|p(z))}_{\text{regularizer by } \alpha} \to \max_{\alpha,\beta}$$

where $p(z)$ is the prior distribution, usually $N(0, \sigma^2 I)$

Reparametrization $q_\alpha(z|x_i):\ z = f(x_i, \alpha, \varepsilon),\ \varepsilon \sim N(0, I)$

$$\sum_{i=1}^{\ell} \underbrace{E_{z\sim q_\alpha(z|x_i)} \log p_\beta(x_i|z)}_{\text{quality reconstruction}} - \underbrace{KL(q_\alpha(z|x_i)\|p(z))}_{\text{regularizer by } \alpha} \to \max_{\alpha,\beta}$$

where $p(z)$ is the prior distribution, usually $N(0, \sigma^2 I)$

Reparametrization $q_\alpha(z|x_i): \ z = f(x_i, \alpha, \varepsilon), \ \varepsilon \sim N(0, I)$

Stochastic gradient method:

- sample $x_i \sim X^\ell, \ \varepsilon \sim N(0, I), \ z = f(x_i, \alpha, \varepsilon)$
- gradient step $\alpha = \alpha + h\nabla_\alpha[\log p_\beta(x_i|f(x_i, \alpha, \varepsilon)) - KL(q_\alpha(z|x_i)\|p(z))]$
  $\beta = \beta + h\nabla_\beta[\log p_\beta(x_i|z)]$

$$\sum_{i=1}^{\ell} \underbrace{E_{z\sim q_\alpha(z|x_i)} \log p_\beta(x_i|z)}_{\text{quality reconstruction}} - \underbrace{KL(q_\alpha(z|x_i)\|p(z))}_{\text{regularizer by } \alpha} \to \max_{\alpha,\beta}$$

where $p(z)$ is the prior distribution, usually $N(0, \sigma^2 I)$

Reparametrization $q_\alpha(z|x_i):\ z = f(x_i, \alpha, \varepsilon),\ \varepsilon \sim N(0, I)$

Stochastic gradient method:

- sample $x_i \sim X^\ell,\ \varepsilon \sim N(0, I),\ z = f(x_i, \alpha, \varepsilon)$
- gradient step $\alpha = \alpha + h\nabla_\alpha[\log p_\beta(x_i|f(x_i, \alpha, \varepsilon)) - KL(q_\alpha(z|x_i)\|p(z))]$
  $\beta = \beta + h\nabla_\beta[\log p_\beta(x_i|z)]$

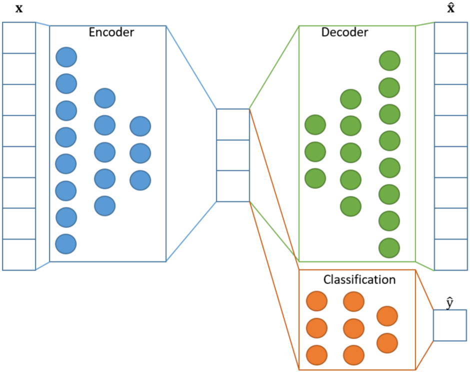
Generation of similar objects:

$$x \sim p_\beta(x|f(x_i, \alpha, \varepsilon)), \varepsilon \sim N(0, I)$$

Autoencoders for supervised learning

Data: unlabeled $(x_i)_{i=1}^{\ell}$, labeled $(x_i, y_i)_{i=\ell+1}^{\ell+k}$

$$z_i = f(x_i, \alpha) \text{ — encoder}$$
$$\hat{x}_i = g(z_i, \beta) \text{ — decoder}$$
$$\hat{y}_i = \hat{y}(z_i, \gamma) \text{ — classifier}$$



Co-learning encoder, decoder and predictive model (classification, regression, etc.)

$$\sum_{i=1}^{\ell} \mathscr{L}(g(f(x_i, \alpha), \beta), x_i) + \lambda \sum_{i=\ell+1}^{\ell+k} \tilde{\mathscr{L}}(\hat{y}(f(x_i, \alpha), \gamma), y_i) \to \min_{\alpha,\beta,\gamma}$$

Loss functions:

$\mathscr{L}(\hat{x}_i, x_i)$ — reconstruction

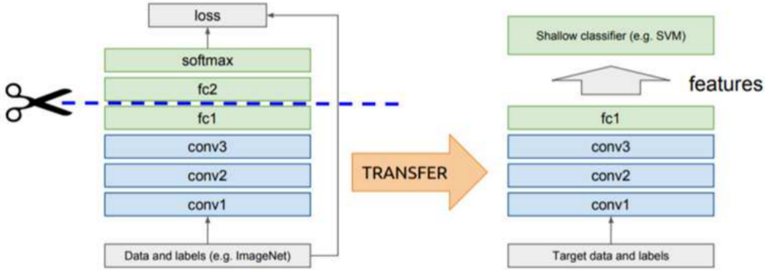
$\tilde{\mathscr{L}}(\hat{y}_i, y_i)$ — prediction

Dor Bank, Noam Koenigstein, Raja Giryes. Autoencoders. 2020.

Pre-training of neural networks

Convolutional network for image processing:

- $z = f(x, \alpha)$ — convolutional layers for object vectorization
- $y = g(z, \beta)$ — fully connected layers for a specific problem



Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson. How transferable are features in deep neural networks? 2014.

Transfer learning

- $f(x, \alpha)$ — universal part of the model (vectorization)
- $g(x, \beta)$ — task-specific part of the model

Basic problem on sample $\{x_i\}_{i=1}^{\ell}$ with loss function $\mathscr{L}_i$:

$$\sum_{i=1}^{\ell} \mathscr{L}_i(f(x_i, \alpha), g(x_i, \beta)) \to \min_{\alpha,\beta}$$

Target problem on another sample $\{x'_i\}_{i=1}^{m}$ with other $\mathscr{L}'_i, g'$:

$$\sum_{i=1}^{m} \mathscr{L}'_i(f(x'_i, \alpha), g'(x'_i, \beta')) \to \min_{\beta'}$$

with $m \ll \ell$ this can be much better than

$$\sum_{i=1}^{m} \mathscr{L}'_i(f(x'_i, \alpha), g'(x'_i, \beta')) \to \min_{\alpha,\beta'}$$

Sinno Jialin Pan, Qiang Yang. A Survey on Transfer Learning. 2009

Multi-task learning

- $f(x, \alpha)$ — universal part of the model (vectorization)
- $g(x, \beta)$ — specific parts of the model for problems $t \in T$

Simultaneous training of the model f on tasks $X_t, t \in T$:

$$\sum_{t \in T} \sum_{i \in X_t} \mathscr{L}_{ti}(f(x_{ti}, \alpha), g(x_{ti}, \beta_t)) \to \min_{\alpha, \{\beta_t\}}$$

Multi-task learning

- $f(x, \alpha)$ — universal part of the model (vectorization)
- $g(x, \beta)$ — specific parts of the model for problems $t \in T$

Simultaneous training of the model f on tasks $X_t, t \in T$:

$$\sum_{t \in T} \sum_{i \in X_t} \mathscr{L}_{ti}(f(x_{ti}, \alpha), g(x_{ti}, \beta_t)) \to \min_{\alpha, \{\beta_t\}}$$

Learnability: the quality of solving a particular problem $\langle X_t, \mathscr{L}_t, g_t \rangle$ improves with increasing sample size $\ell_t = |X_T|$.

Multi-task learning

- $f(x, \alpha)$ — universal part of the model (vectorization)
- $g(x, \beta)$ — specific parts of the model for problems $t \in T$

Simultaneous training of the model f on tasks $X_t, t \in T$:

$$\sum_{t \in T} \sum_{i \in X_t} \mathscr{L}_{ti}(f(x_{ti}, \alpha), g(x_{ti}, \beta_t)) \to \min_{\alpha, \{\beta_t\}}$$

Learnability: the quality of solving a particular problem $\langle X_t, \mathscr{L}_t, g_t \rangle$ improves with increasing sample size $\ell_t = |X_T|$.

Learning to learn: the quality of the solution of each of the problems $t \in T$ improves with the growth of ℓ_t and the total number of problems $|T|$.

Multi-task learning

- $f(x, \alpha)$ — universal part of the model (vectorization)
- $g(x, \beta)$ — specific parts of the model for problems $t \in T$

Simultaneous training of the model f on tasks $X_t, t \in T$:

$$\sum_{t \in T} \sum_{i \in X_t} \mathscr{L}_{ti}(f(x_{ti}, \alpha), g(x_{ti}, \beta_t)) \to \min_{\alpha, \{\beta_t\}}$$

Learnability: the quality of solving a particular problem $\langle X_t, \mathscr{L}_t, g_t \rangle$ improves with increasing sample size $\ell_t = |X_T|$.

Learning to learn: the quality of the solution of each of the problems $t \in T$ improves with the growth of ℓ_t and the total number of problems $|T|$.

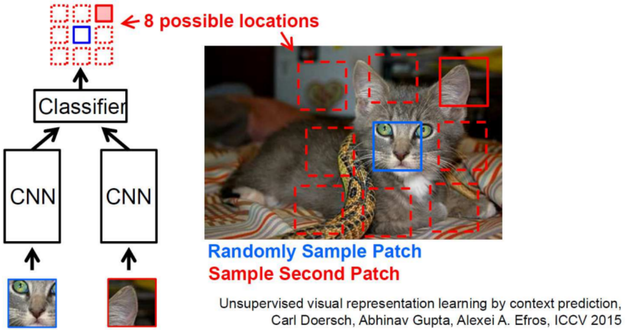
Few-shot learning: a small number of examples, sometimes even one, is enough to solve the t problem.

M.Crawshaw. Multi-task learning with deep neural networks: a survey. 2020

Y.Wang et al. Generalizing from a few examples: a survey on few-shot learning. 2020

Self-supervised learning

The vectorization model $z = f(x, \alpha)$ is trained to predict the relative positions of pairs of fragments of the same image



Benefit: the network learns vector representations of objects without a labeled training set. Their quality is not less than obtained by labeled ImageNet.

Model distillation or surrogate modeling

Training complex model $a(x, w)$ is "long time, expensive":

$$\sum_{i=1}^{\ell} \mathscr{L}(a(x_i, w), y_i) \to \min_{w}$$

Training simple model $b(x, w')$, possibly on other data:

$$\sum_{i=1}^{k} \mathscr{L}(b(x'_i, w'), a(x'_i, w)) \to \min_{w'}$$

Model distillation or surrogate modeling

Training complex model $a(x, w)$ is "long time, expensive":

$$\sum_{i=1}^{\ell} \mathscr{L}(a(x_i, w), y_i) \to \min_{w}$$

Training simple model $b(x, w')$, possibly on other data:

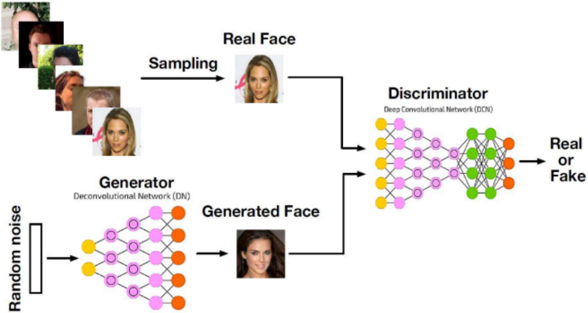
$$\sum_{i=1}^{k} \mathscr{L}(b(x'_i, w'), a(x'_i, w)) \to \min_{w'}$$

Problem examples:

- replacement of a complex model (climate, aerodynamics, etc.), which takes months to calculate on a supercomputer, with an "light" approximating surrogate model
- replacement of a complex neural network, which is trained for weeks on big data, with an "light" approximating neural network with minimization of the number of neurons and connections

Generative Adversarial Net, (GAN)

The generator $G(z)$ learns to generate objects x from noise z
The discriminator $D(x)$ learns to distinguish them from real objects



Antonia Creswell et al. Generative Adversarial Networks: an overview. 2017.
Zhengwei Wang, Qi She, Tomas Ward. Generative Adversarial Networks: a survey and taxonomy. 2019.

Chris Nicholson. A Beginner's Guide to Generative Adversarial Networks. 2019.

GAN problem statement

There is a selection of objects $\{x_i\}_{i=1}^m$ from X
We train

- probabilistic generative model $G(z, \alpha) : x \sim p(x|z, \alpha)$
- probabilistic discriminative model $D(x, \beta) = p(1|x, \beta)$

GAN problem statement

There is a selection of objects $\{x_i\}_{i=1}^{m}$ from X
We train

- probabilistic generative model $G(z, \alpha) : x \sim p(x|z, \alpha)$
- probabilistic discriminative model $D(x, \beta) = p(1|x, \beta)$

Criteria:

- Discriminative model training D:

$$\sum_{i=1}^{m} \ln D(x_i, \beta) + \ln(1 - D(G(z_i, \alpha), \beta)) \to \max_{\beta}$$

- learning the generative model G from random noise $\{z_i\}_{i=1}^{m}$:

$$\sum_{i=1}^{m} \ln(1 - D(G(z_i, \alpha), \beta)) \to \min\ limits_{\alpha}$$

Ian Goodfellow et al. Generative Adversarial Nets. 2014

StyleGAN demo

Let's watch the video

Papers are here: https://nvlabs.github.io/stylegan2/versions.html

Summary

- Clustering and Kohonen maps
- Autoencoders including co-learning with classification
- Multi-task learning
- Transfer learning
- Distillation and surrogate modeling
- Adversarial networks (GANs)

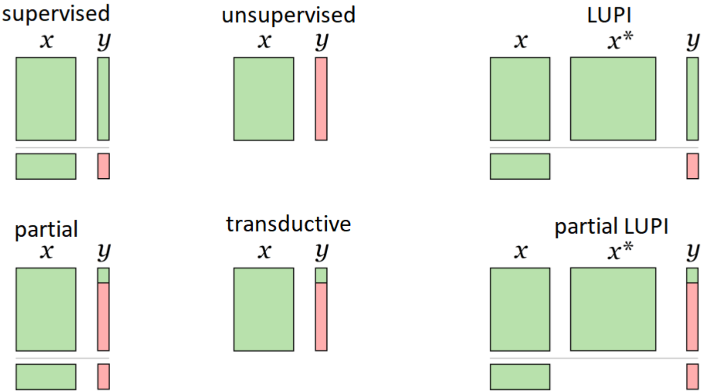
Summary

- Clustering and Kohonen maps
- Autoencoders including co-learning with classification
- Multi-task learning
- Transfer learning
- Distillation and surrogate modeling
- Adversarial networks (GANs)

What else can you see?

- Mini-course by S.I. Nikolenko about GANs from three lectures (in Russian)
- Lec. 13 of the CS231n GAN course

Learning Using Privileged Information (LUPI)



V. Vapnik, A. Vashist. A new learning paradigm: Learning Using Privileged Information // Neural Networks. 2009.

Examples of problems with privileged information x^*

- x — primary (1D) protein structure
  x^* — tertiary (3D) protein structure
  y — hierarchical classification of protein function
- x — history of the time series
  x^* — information about the future behavior of the series
  y — forecast of the next point of the series
- x — text document
  x^* — highlighted keywords or phrases
  y — document category
- x — pair (request, document)
  x^* — keywords or phrases highlighted by the assessor
  y — relevance score

Problem of Learning with Privileged Information

- Separate training of student model and teacher model:

$$\sum_{i=1}^{\ell} \mathscr{L}(a(x_i, w), y_i) \to \min_w \qquad \sum_{i=1}^{\ell} \mathscr{L}(a(x_i^*, w^*), y_i) \to \min_w$$

- The student model learns to repeat the mistakes of the teacher model:

$$\sum_{i=1}^{\ell} \mathscr{L}(a(x_i, w), y_i) + \mu \mathscr{L}(a(x_i, w), a(x_i^*, w^*)) \to \min_w$$

- **Co-learning** student model and teacher model:

$$\sum_{i=1}^{\ell} \mathscr{L}(a(x_i, w), y_i) +$$
$$+ \lambda \mathscr{L}(a(x_i^*, w^*), y_i) + \mu \mathscr{L}(a(x_i, w), a(x_i^*, w^*)) \to \min_{w, w^*}$$

D.Lopez-Paz, L.Bottou, B.Scholkopf, V.Vapnik. Unifying distillation and privileged information. 2016.