The Mathematics of Deep Learning

Sjoerd Verduyn Lunel (Utrecht University)

Introduction

The Deep Learning textbook by Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016 is an excellent resource intended to help students and practitioners enter the field of machine learning in general and deep learning in particular.

The online version of the book is available online for free.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

I will not cover all material in the notes and during the werkcollege it is possible to concentrate on

- Refreshing Differential Calculus
- Refreshing Linear Algebra
- Simulating Neural Networks in a Python environment

Prelude: Machine learning: what would we like to predict

Classification. Consider a set of images of dogs,plants and houses.

Task Map an image to scores of a label, i.e., a vector $(.8, .1, .1)$ indicates 80% it is a dog, 10% it is a plant and 10 % it is a house.

Data representation An image consists pixels with a number between 0 ($=$ black) and 255 ($=$ white) at each pixel. A color picture is similar with three color channels overlayed. For example, a color picture of 248 times 400 pixels consists of

$$248 \times 400 \times 3 = 297600 \text{ numbers between } 0 \text{ and } 255 \text{ each.}$$

A flattened image is a vector in $\mathbb{R}^p$ (Here $p = 297600$).

We can compare images by comparing the (metric) distance in $\mathbb{R}^p$.

A set of images can be represented by a cloud of points in $\mathbb{R}^p$.

Prelude: Machine learning II

Training set
$$T = \{x^{(i)} \in \mathbb{R}^p,\ y^{(i)} \in \{1,2,3\} \mid 1 \le i \le N\}.$$
Note that in this representation the score vector of $x^{(i)}$ is a vector with all zeros except for a 1 at the $y^{(i)}$-coordinate.

Challenge How to create a rule

$$F : \mathbb{R}^p \to \mathbb{R}^3$$

that maps a given image to a score vector.

Model Architecture Consider an affine classifier

$$F(x) = Wx + b$$

where W is a $3 \times p$-matrix and b is 3-vector.

Prelude: Machine learning III

For example if $p = 4$ we could have for $F(x) = Wx + b$

$$F(x) = \begin{pmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0.0 & 0.25 & 0.2 & -0.3 \end{pmatrix} \begin{pmatrix} 56 \\ 231 \\ 24 \\ 2 \end{pmatrix} + \begin{pmatrix} 1.1 \\ 3.2 \\ -1.2 \end{pmatrix} = \begin{pmatrix} -96.8 \\ 437.9 \\ 61.95 \end{pmatrix}$$

with $x = (56, 231, 24, 2)^T$ a flattened image with four pixels.

Recall that matrix multiplication of a matrix and a vector corresponds to taking the inner product of a row of the matrix and the vector.

Therefore every row of the matrix W is a classifier (template) for one of the classes

Template matching taking the inner product of a template and a flattened image.

Prelude: Machine learning IV

Learning Optimization of W and b by minimizing some loss or cost function on the training set T. For example

$$C : \mathbb{R}^{3p+3} \to \mathbb{R}$$

given by MSD (= Mean Square Distance)

$$C(W,b) = \sum_{i=1}^{m} \left\| F(x^{(i)}) - e_{y^{(i)}} \right\|^2$$

where $m \leq N$. Later we will see other loss functions as well.

Example - Decision Function

Consider the data

CD	Rating 1	Rating 2	I like
1	1	4	No
2	1	5	No
3	1.5	4	No
4	2.5	1	No
5	2.5	1.5	No
6	2.5	3	No
7	2.5	5	Yes
8	3.5	4	Yes
9	3.5	5	Yes
10	4.5	4	Yes
11	4.5	5	Yes

Task Do I like a new CD with Rating 1 $=$ Rating 2 $= 3$.

Example - Decision Function II

Model Architecture: Bounded decision function $F : \mathbb{R}^2 \to \mathbb{R}$

$$F(x) = f(Wx + b), \quad W = \begin{pmatrix} w_{11} & w_{12} \end{pmatrix}, \ b \in \mathbb{R}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a given function.

For example, we can take the Sigmoid Function

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Analyse the data and the model architecture and derive a differential equation for f.

Training set:

$$T = \left\{ x^{(i)} \in \mathbb{R}^2, \ y^{(i)} \in \{0, 1\} \mid 1 \leq i \leq 11 \right\}$$

So, for example,

$$x^{(1)} = (1, 4) \text{ and } y^{(1)} = 0, \qquad x^{(11)} = (4.5, 5) \text{ and } y^{(11)} = 1$$

Example - Decision Function III

Learning/Training: the decision function by minimizing a cost function $C : \mathbb{R}^3 \to \mathbb{R}$

$$C(w_{11}, w_{12}, b) = \sum_{i=1}^{m} \big(F(x^{(i)}) - y^{(i)}\big)^2 \qquad (m \leq 11).$$

Compute the partial derivatives of the loss function C with respect to w_{11}, w_{12}, b

$$\begin{aligned}
\frac{\partial}{\partial w_{11}} \big(F(x^{(i)}) - y^{(i)}\big)^2 &= 2\big(F(x^{(i)}) - y^{(i)}\big) \frac{\partial}{\partial w_{11}} F(x^{(i)}) \\
&= 2\big(F(x^{(i)}) - y^{(i)}\big) \frac{\partial}{\partial w_{11}} f(w_{11} x_1^{(i)} + w_{12} x_2^{(i)} + b) \\
&= 2\big(F(x^{(i)}) - y^{(i)}\big) f'(w_{11} x_1^{(i)} + w_{12} x_2^{(i)} + b) x_1^{(i)}
\end{aligned}$$

Similarly we can compute the other derivatives and the gradient ∇C of C.

Example - Decision Function IV

Stochastic Gradient Descend (SGD)

1. initialize w_{11}, w_{12} and b at random
2. pick a random example from the training set $x^{(i)}, y^{(i)}$ and take $m = 1$ in the definition of the loss function
3. compute the gradient ∇C of the loss function C
4. update w_{11}, w_{12} and b by

$$
\begin{aligned}
w_{11} &:= w_{11} - \alpha \frac{\partial C}{\partial w_{11}} \\
w_{12} &:= w_{12} - \alpha \frac{\partial C}{\partial w_{12}} \\
b &:= b - \alpha \frac{\partial C}{\partial b}
\end{aligned}
$$

5. Go back to step 2

Example - Decision Function V

After we have learned the decision function by minimizing the loss function we can compute the decision boundary function.

The Decision Boundary are the values of x that give

$$\{x \in \mathbb{R}^2 \mid F(x) = 1/2\}$$

Now add data

CD	Rating 1	Rating 2	I like
12	1	1	Yes
13	1	1.5	Yes
14	1.5	1	Yes
15	5	1.5	Yes

and plot the complete data set.

What can you say about the decision boundary function.

Intermezzo: Differential Calculus - Linear Maps

A map $A : \mathbb{R}^n \to \mathbb{R}^m$ is called a linear transformation (linear operator) if

$$A(x + y) = A(x) + A(y), \quad A(\lambda x) = \lambda A(x)$$

for every $x, y \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$.

The standard matrix (a_{ij}) of A is the matrix given with respect to the standard basis $e_1, \dots, e_n$ and $\tilde{e}_1, \dots, \tilde{e}_m$ in, respectively, $\mathbb{R}^n$ and $\mathbb{R}^m$.

Note that the elements of the standard matrix are given by inner products

$$a_{ij} = \langle Ae_j, \tilde{e}_i \rangle$$

and hence the j-th column of the matrix (a_{ij}) equals Ae_j.

If A is symmetric $A = A^T$, then there exists an orthonormal bases of eigenvectors such that the matrix representation of A with respect to this basis becomes diagonal.

Intermezzo: Differential Calculus - Differentiation

Recall that a function $f : \mathbb{R} \to \mathbb{R}$ is differentiable in x if the limit

$$A = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

exists. The value A we denote by $f'(x)$.

In order to adapt this definition to functions of more variables, we have to reformulate this definition: a function f is differentiable in x if there exists a real A such that

$$\lim_{h \to 0} \frac{|f(x+h) - f(x) - Ah|}{|h|} = 0.$$

Let $E \subset \mathbb{R}^n$ be an open set and consider a function $f : E \to \mathbb{R}^m$ in n variables with values in $\mathbb{R}^m$.

We call f differentiable in x if there exists a linear transformation $A : \mathbb{R}^n \to \mathbb{R}^m$ such that

$$\lim_{h \to 0} \frac{\| f(x+h) - f(x) - Ah \|}{\|h\|} = 0$$

Note that h now denotes a vector in $\mathbb{R}^n$. We write $A = f'(x) = Df(x)$

Intermezzo: Differential Calculus - Differentiation II

Check that a linear transformation $F : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable on $\mathbb{R}^n$ and $F'(x) = F$ for every $x \in \mathbb{R}^n$.

Consider the function $F : \mathbb{R}^3 \to \mathbb{R}^2$ defined by

$$F(x, y, z) = \begin{pmatrix} x^3 + y^2 \\ y + z \end{pmatrix}$$

Note that

$$F(x + h, y + k, z + l) - f(x, y, z) = \begin{pmatrix} 3x^2h + 3xh^2 + h^3 + 2yk + k^2 \\ k + l \end{pmatrix} =$$

$$= \begin{pmatrix} 3x^2 & 2y & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} h \\ k \\ l \end{pmatrix} + \begin{pmatrix} 3xh^2 + h^3 + k^2 \\ 0 \end{pmatrix}$$

Check that we can define

$$A = \begin{pmatrix} 3x^2 & 2y & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

and that F is differentiable in (x, y, z) with derivative A.

Intermezzo: Differential Calculus - Directional derivative

In order to derive a matrix representation of the derivative, we first consider directional derivaties.

Let $E \subset \mathbb{R}^n$ and $F : E \to \mathbb{R}^m$ be differentiable with derivative $DF(x)$.

Check that if $v \in \mathbb{R}^n$ with $\|v\| = 1$, then

$$\lim_{t \to 0} \frac{F(x+tv) - F(x)}{t} = DF(x)v.$$

We can define the directional derivative of F in direction v by

$$D_vF(x) := \lim_{t \to 0} \frac{F(x+tv) - F(x)}{t}$$

and

$$\frac{\partial F(x)}{\partial x_j} = D_jF(x) := D_{e_j}F(x).$$

If F is differentiable in x, then the standard matrix of $DF(x)$ is called the Jacobian of F at x.

Intermezzo: Differential Calculus - The chain rule

Let $E \subset \mathbb{R}^n$ and $D \subset \mathbb{R}^m$ open sets and $f : E \to \mathbb{R}^m$ and $g : D \to \mathbb{R}^k$ given functions. We assume that $f[E] \subset D$ so that the composition $F := g \circ f : E \to \mathbb{R}^k$ is well defined.

Chain rule If f is differentiable in $x_0 \in E$ and g is differentiable in $y_0 = f(x_0)$, then is the composition $F = g \circ f$ differentiable in x_0 and

$$F'(x_0) = g'(y_0) \circ f'(x_0).$$

The chain rule allows us to express the partial derivatives of $D_j F_i(x_0)$ in terms of the partial derivatives of g and f and we have (check) that

$$D_j F_i(x_0) = \sum_{\nu=1}^{m} D_\nu g_i(y_0) D_j f_\nu(x_0),$$

or

$$\frac{\partial F_i}{\partial x_j}(x_0) = \sum_{\nu=1}^{m} \frac{\partial g_i}{\partial y_\nu}(y_0) \frac{\partial f_\nu}{\partial x_j}(x_0).$$

Intermezzo: Differential Calculus - The gradient

Let $F : \mathbb{R}^n \to \mathbb{R}$. Suppose that F is differentiable in x. By definition the *gradient* of F in x equals the vector

$$\nabla F(x) = J_F(x)^T = (D_1 F(x), \ldots, D_n F(x))^T.$$

Check that If F is differentiable in x, then

$$F'(x)h = \langle \nabla f(x), h \rangle, \quad h \in \mathbb{R}^n.$$

Note that the directional derivative $D_v F(x)$ is maximal if v (with $\|v\| = 1$) is in the same direction as $\nabla F(x)$, indeed

$$\langle \nabla f(x), v \rangle = \|\nabla f(x)\| \cdot \|v\| \cos \varphi,$$

where φ denotes the angle between the vectors $\nabla F(x)$ and v.

Thus the gradient of F in x points in the direction of the maximal increase of the function seen from x.

A more complex classification problem

Classification Consider a set of images of digits $0, 1, 2, \ldots, 9$ and recall the data representation of images.

Task Map images to scores of a label $F : \mathbb{R}^p \to \mathbb{R}^{10}$.

Model architecture The model can no longer be linear, images of digits do not add up well, for example, two images of zeros could become an image of the digit eight.

What is the right class of functions to consider?

New idea 1: Consider continuous piecewise linear functions (CPL):

$$F(x) = W_2 \max\{0, W_1 x + b_1\} + b_2$$

where W_1 is a $q \times p$-matrix, $b_1 \in \mathbb{R}^q$ and W_2 is a $10 \times q$-matrix, $b_2 \in \mathbb{R}^{10}$.

This class of functions is already very large and has the universal approximation property, but is not yet large enough.

New idea 2: The best way to create complex functions from simple functions is by composition of functions.

Example - continuous piecewise linear functions

Consider the XOR function defined by

$$\text{XOR}(0,0)=0,\ \text{XOR}(1,0)=1,\ \text{XOR}(0,1)=1,\ \text{XOR}(1,1)=0$$

and train a network on these four points in the training set

Model architecture

$$F(x_1,x_2)=w_1x_1+w_2x_2+b$$

Cost function (MSE)

$$\begin{aligned}C(w_1,w_2,b)&=\frac{1}{4}\sum_{x\in T}\big(y-F(x)\big)^2\\&=\frac{1}{4}\big(b^2+(w_1+b-1)^2+(w_2+b-1)^2+(w_1+w_2+b)^2\big)\end{aligned}$$

Compute the gradient of C and the minimum of C. This gives $w_1=w_2=0$ and $b=1/2$.

So a affine model is not capable to represent the XOR function.

Example - continuous piecewise linear functions II

Next consider a composition of two functions

Model architecture

$$F(z) = W \max\{0, wx + b\} + c$$

Take

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \; w = \begin{pmatrix} 1 & -2 \end{pmatrix}, \; c = 0.$$

and check that F is given by

$$\begin{aligned} F(x, w, b, W, c) &= \begin{pmatrix} 1 & -2 \end{pmatrix} \max\{0, \begin{pmatrix} x_1 + x_2 \\ x_1 + x_2 - 1 \end{pmatrix}\} \\ &= \begin{pmatrix} 1 & -2 \end{pmatrix} \begin{pmatrix} x_1 + x_2 \\ \max\{0, x_1 + x_2 - 1\} \end{pmatrix} \\ &= x_1 + x_2 - 2 \max\{0, x_1 + x_2 - 1\} \end{aligned}$$

and models the XOR function.

Give a network interpretation of the function F.

Summary - continuous piecewise linear functions

Continuous piecewise linear functions can model more complicated score functions

Two layers

$$F(x) = W_2 \max\{0, W_1 x\}$$

(how can we include the bias b in the matrix W_1?)

Three layers

$$F(x) = W_3 \max\{0, W_2 \max\{0, W_1 x\}\}$$

Sizes of the matrices W_1, W_2 and W_3 are called hyperparameters

Cross validation Divide the training set into 5 equal batches, use four of them for training and one for validation, then iterate over which fold is the validation fold, evaluate performance and update hyperparameters, and average over the different folds

Loss function/Cost function

$$T = \{(x^{(i)}, y^{(i)}) \mid 1 \leq i \leq N\}, \qquad C_0(x^{(i)}) = L(x^{(i)}, y^{(i)})$$

and

$$B \subset \{x^{(1)}, \ldots, x^{(N)}\}, \qquad C_0(B) = \frac{1}{|B|} \sum_{x \in B} C_0(x)$$

Summary - Stochastic Gradient Descend (mini-batch)

Divide the training set in equal batches at random, take a batch B_i and compute

$$C_0(B_i) = \frac{1}{|B|} \sum_{x \in B} C_0(x)$$

and the gradient $\nabla C_0(B_i)$ (a vector of the same size as the number of parameters (weights) in the model (network)).

After completing a batch we update the all weights by minus a small multiple of $\nabla C_0(B_i)$.

Note that $\nabla C_0(B_i)$ is an estimate for $\nabla C_0(T)$.

After we have used all batches in the training set we have completed one epoch.

We can repeat this process for many epochs.

Typically we define the dataset into three sets: training, validation and testing. We run through the training set for many epochs and after every epoch we update the hyperparameters.

Summary - Loss functions

Let $T = \{(x^{(i)}, y^{(i)}) \mid 1 \leq i \leq N\}$ a training set.

Loss function/Cost function: Support Vector Machine (SVM) Let $s_j = F(x^{(i)})_j$ a score vector and

$$L_i = \sum_{j \neq y^{(i)}} \max\{0, s_j - s_{y^{(i)}} + \Delta\}.$$

For example, let $s = [13, -7, 11]^T$ the score vector and $y^{(i)} = 1$. Take $\Delta = 10$.

Then

$$L_i = \max\{0, -7 - 13 + 10\} + \max\{0, 11 - 13 + 10\} = 8$$

The first term shows no data loss. Although the score function is highest at the correct index it is not by sufficient margin Δ (a hyperparameter).

Regularisation Weights are not unique, put penalty on large weights in the loss function

$$L = \frac{1}{N} \sum_i L_i + \lambda \sum_{k,l} W_{kl}^2$$

The first term models data loss and the second regularisation loss.

Feed forward neural network - the bookkeeping

Suppose there are L layers

$$F = F_L \circ F_{L-1} \circ \cdots \circ F_1$$

A node j in layer l receives an input $z_j^{(l)}$ and sends an activation output $a_j^{(l)} = f^{(l)}(z_j^{(l)})$ for $1 \leq l \leq L$ and the output of the network is $a^{(L)}$.

The model architecture is given in matrix terms by

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$
$$a_j^{(l)} = f^{(l)}(z_j^{(l)})$$

where $f^{(l)}$ is activation function (ReLu or Sigmoid) of layer l.

The output of layer l defines the function F_l by

$$F_l(a^{(l-1)}) = a^{(l)}$$

Feed forward neural network - back propagation

Let $T = \{(x^{(i)}, y^{(i)}) \mid 1 \leq i \leq N\}$ a training set and $B \subset \{x^{(1)}, \dots, x^{(N)}\}$ a (random) batch of training input.

Let C_0 be a given loss function

$$C_0(B) = \frac{1}{2|B|} \sum_{x \in B} \|F(x) - y(x)\|^2$$

where $y(x)$ denotes the desired output of the network when the input is $x \in B$.

Compute the gradient of $C_0(B)$ by using the chain rule.

$$\begin{aligned} \frac{\partial C_0(B)}{\partial w_{jk}^{(L)}} &= \frac{\partial C_0(B)}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \\ &= (F_j(x) - y_j(x)) f^{(L)\prime}(z_j^{(L)}) a_k^{(L-1)} \end{aligned}$$

Feed forward neural network - back propagation II

Similarly for $l < L$

$$\frac{\partial C_0(B)}{\partial w_{jk}^{(l)}} = \frac{\partial C_0(B)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}$$

$$= \frac{\partial C_0(B)}{\partial a_j^{(l)}} f^{(l)\prime}(z_j^{(l)}) a_k^{(l-1)}$$

To compute

$$\frac{\partial C_0(B)}{\partial a_j^{(l)}}$$

for $l < L$, we can use recursion and assume that

$$\frac{\partial C_0(B)}{\partial a_j^{(l+1)}} \quad \text{is known}$$

Feed forward neural network - back propogation II

From the chain rule and the model architecture

$$\begin{aligned} z^{(l)} &= W^{(l)} a^{(l-1)} + b^{(l)} \\ a_j^{(l)} &= f^{(l)}(z_j^{(l)}) \end{aligned}$$

we obtain

$$\begin{aligned} \frac{\partial C_0(B)}{\partial a_j^{(l)}} &= \sum_i \frac{\partial C_0(B)}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial a_j^{(l)}} \\ &= \sum_i \frac{\partial C_0(B)}{\partial a_i^{(l+1)}} f^{(l+1)\prime}(z_i^{(l+1)}) w_{ij}^{(l+1)}. \end{aligned}$$

This way we can compute the complete gradient and update the weights

$$W := W - \eta \nabla C_0(B).$$

Singular Value Decomposition (SVD) - Data representation

Let A be a $m \times n$-matrix of rank r.

Let Σ be $m \times n$-matrix such that

$$\Sigma = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$$

with

$$D = \mathsf{diag}\,(\sigma_1, \sigma_2, \ldots, \sigma_r), \qquad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$$

where $\sigma_i = \sqrt{\lambda_i}$ with λ_i the eigenvalues of $A^T A$.

There exist orthogonal matrices U $(m \times m)$ and V $(n \times n)$ such that

$$A = U \Sigma V^T$$

where $V = [v_1 v_2 \cdots v_n]$ with v_i an orthogonal basis of eigenvectors of $A^T A$.

Note

$$0 \leq \|Av_i\|^2 = \left(Av_i\right)^T Av_i = v_i^T A^T A v_i = v_i^T \lambda_i v_i = \lambda_i$$

Singular Value Decomposition (SVD) - Example

Let A be a 3×2 matrix given by

$$A = \begin{pmatrix} 1 & -1 \\ -2 & 2 \\ 2 & -2 \end{pmatrix}$$

and compute the SVD of A.

Compute the eigenvalues and eigenvectors of $A^T A$

$$A^T A = \begin{pmatrix} 9 & -9 \\ -9 & 9 \end{pmatrix}$$

and

$$\det(A^T A - \lambda I) = (\lambda - 9)^2 - 81 = \lambda(\lambda - 18)$$

At $\lambda = 18$

$$v_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \end{pmatrix}$$

At $\lambda = 0$

$$v_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \end{pmatrix}$$

Singular Value Decomposition (SVD) - Example II

So

$$V = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \qquad \Sigma = \begin{pmatrix}3\sqrt{2} & 0\\ 0 & 0\\ 0 & 0\end{pmatrix}$$

To find U note that $AV = U\Sigma$ and

$$u_1 = \frac{1}{3\sqrt{2}}Av_1 = \frac{1}{3}\begin{pmatrix}1\\ -2\\ 2\end{pmatrix}$$

To find u_2 and u_3 we have to find an orthogonal basis of the plane othogonal to u_1

$$x_1 - 2x_2 + 2x_3 = 0.$$

This yields

$$u_1 = \frac{1}{3}\begin{pmatrix}1\\ -2\\ 2\end{pmatrix},\ u_2 = \frac{1}{\sqrt{5}}\begin{pmatrix}2\\ 1\\ 0\end{pmatrix},\ u_3 = \frac{1}{\sqrt{45}}\begin{pmatrix}-2\\ 4\\ 5\end{pmatrix}$$

Singular Value Decomposition (SVD) - Example II

Let A be a 2×3 matrix given by

$$A = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix}$$

and compute the SVD of A.

Answer:

$$A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{pmatrix}$$

Applications of SVD to image processing and statistics

Mean and Covariance

$$X = [x^{(1)} \cdots x^{(N)}] \quad p \times N \text{ matrix of observations}$$

Define

$$m = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}, \qquad y^{(i)} = x^{(i)} - m$$

and the mean-deviation form

$$Y = [y^{(1)} \cdots y^{(N)}]$$

The covariance matrix is given by

$$S = \frac{1}{N-1} Y Y^T \quad p \times p \text{ matrix}$$

Here s_{jj} is the variance of $x_j^{(i)}$, $1 \leq j \leq N$ and s_{kj} is the covariance of $x_k^{(i)}$ and $x_j^{(i)}$, $1 \leq k \leq N$ for $k \neq j$.

Applications of SVD to image processing and statistics II

Find the mean and covariance of the observation data

$$x^{(1)} = \begin{pmatrix}1\\2\\1\end{pmatrix},\ x^{(2)} = \begin{pmatrix}4\\2\\13\end{pmatrix},\ x^{(3)} = \begin{pmatrix}7\\8\\1\end{pmatrix},\ x^{(4)} = \begin{pmatrix}8\\4\\5\end{pmatrix}$$

Idea Transform the observation data X with mean zero and covariance matrix S

$$x^{(i)} = Uy^{(i)}, \qquad 1 \le i \le N$$

by an orthogonal transformation

$$y^{(i)} = U^{-1}x^{(i)} = U^T x^{(i)}$$

such that the covariance matrix of the transformed data

$$Y = [y^{(1)} \cdots y^{(N)}]$$

given by $U^T SU$ becomes diagonal.

Applications of SVD to image processing and statistics III

In other words we need to find U such that

$$U^T S U = D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p), \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0.$$

The eigenvectors corresponsing to λ_i, $1 \leq i \leq p$, are called the principal components of the data

Connection SVD If X is a $p \times N$ matrix of observations in mean-deviation form and if

$$A = \frac{1}{\sqrt{N-1}} X^T,$$

then $A^T A$ is the covariance matrix of X.

So the squares of the singular values of A are exaxtly the p eigenvalues of the covariance matrix S of X and the right singular vectors of A are the principal components of the observation data X.

Example

$$X = \begin{pmatrix} 14 & 22 & 6 & 3 & 2 & 20 \\ 12 & 6 & 9 & 15 & 13 & 5 \end{pmatrix}$$

Find the mean-deviation form and the sample covariance matrix.

Face recognition using eigenfaces

Let X be a $p \times N$ matrix consisting of N flattened images

$$X = [x^{(1)} \cdots x^{(N)}], \qquad x^{(i)} \in \mathbb{R}^p$$

Let

$$m = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}, \quad y^{(i)} = x^{(i)} - m$$

Let

$$Y = [y^{(1)} \cdots x^{(N)}], \qquad y^{(i)} \in \mathbb{R}^p,$$

the data in mean-deviation form and

$$u_i, \quad 1 \leq i \leq p, \quad \text{the principal components of the data}$$

Let M be the number of significant singular values and given a new image z we project the image onto the significant principal components

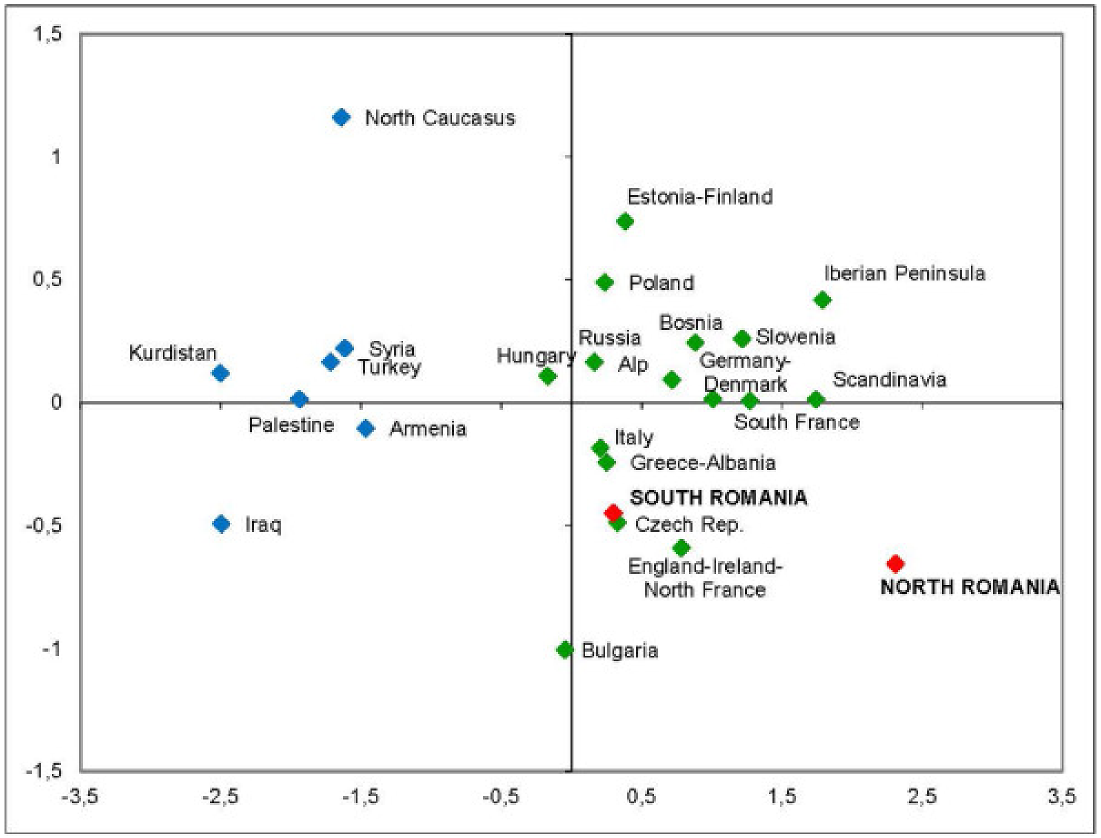
$$z_k = u_k^T (z - m), \qquad 1 \leq k \leq M$$

and the vector $z = (z_1, z_2, \ldots, z_M)^T$ describes the contribution of each eigenface in representing the new image of a face.

Next apply the classification theory developed in this course to (transformed) data.

Representing the data - Distance matrices

Multidimensional scaling (MDS) techniques are used to model distance data, for example, from classification, as points in a Euclidean space B.



Figuur 1: Genetic distances, calculated according to the distribution of the mtDNA haplogroup frequencies

Multidimensional scaling (MDS)

Given a distance matrix M ($M = M^T$ with zeros on the diagonal) find vectors

$$Y = [y_1, y_2, \ldots, y_n]$$

in $\mathbb{R}^q$ such that

$$m_{ij} = \|y_i - y_j\|$$

Note that the solution is not unique and therefore we assume in addition that

$$\sum_{i=1}^{n} Y_{ik} = 0 \quad \text{voor alle } 1 \leq k \leq n \tag{1}$$

Idea: Let $B = YY^T$ then because of the relations between norm and inner product we have

$$m_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij} \tag{2}$$

From (1) we derive the relation

$$\sum_{i=1}^{n} b_{ij} = 0$$

Multidimensional scaling (MDS) II

Therefore

$$\sum_{i=1}^{n} m_{ij}^2 = \mathsf{trace}(B) + n b_{jj} \quad \text{en} \quad \sum_{j=1}^{n} m_{ij}^2 = \mathsf{trace}(B) + n b_{ii}$$

and

$$\sum_{j=1}^{n} \sum_{i=1}^{n} m_{ij}^2 = 2n\, \mathsf{trace}(B)$$

This shows that we can solve for b_{ij}

$$b_{ij} = -\frac{1}{2}\Big(m_{ij}^2 - \frac{1}{n}\sum_{j=1}^{n} m_{ij}^2 - \frac{1}{n}\sum_{i=1}^{n} m_{ij}^2 + \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n} m_{ij}^2 \Big)$$

This yields Y $(B = YY^T)$ and using the eigenvalue decomposition of B:

$$B = V \Lambda V^T$$

we arrive at

$$Y = \Lambda^{1/2} V^T.$$

How to install Python and Keras

Ubuntu

Ubuntu is a Linux environment for Windows 10 which can be downloaded from the Microsoft App Store

In Ubuntu 18.04 Python 3.6 should be pre-installed.

If the Ubuntu package list not updated for a while, type

```
sudo apt update
```

Python

If pip happens to be not installed in Ubuntu (18.04), type

```
sudo apt install python3-pip
```

To check if you have the correct Python installation, type

```
pip3 --version
```

This should return

```
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6
```

How to install Python and Keras II

Jupyter

It is convenient to use Jupyter Notebooks when programming with Python and Keras.

Jupyter can be installed as follows

```
    pip3 install jupyterlab
```

Running Jupyter

To generate a browser link for Jupyter first go to the desired folder on your computer (for example, the Documents folder) and start Jupyter as follows

```
    cd /mnt/c/Users/User/Documents
    jupyter notebook
```

How to install Python and Keras III

Keras

To Keras we first have to install Tensorflow backend using pip3

```
pip3 install tensorflow
```

Next we install Keras using pip3

```
pip3 install keras
```

Python packages

If Python packages such as numpy, scipy or matplotlib is not preinstalled, you can install them using pip3

```
pip3 install numpy
pip3 install scipy
pip3 install matplotlib
```

Keras - Set up a neural network in Python

Define functions and classes

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

Data set (csv file): music.csv

The data set consists of rows

```
1,4,0
1,5,0
1.5,4,0
....
4.5,5,1
```

Load the data set

```
dataset = loadtxt('music.csv', delimiter=',')
#split the data into input (x) and output (y) variables
x = dataset[:,0:2]
y = dataset[:,2]
```

Keras - Set up a neural network in Python II

Define the model architecture

```
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Compile the model

```
model.compile(loss='mean_squared_error', optimizer='Adam',
metrics=['accuracy'])
```

Train/Learn the model

```
model.fit(x, y, epochs = 15, batch_size=2)
```

Evaluate the model on the training set

```
predictions = model.predict_classes(x)
for i in range(len(X)):
        print('%s => %d (expected %d)' % (x[i].tolist(),
                                            predictions[i], y[i]))
```

Keras - Set up a neural network in Python III

Make predictions with the network on new data

```
model.predict( np.reshape( np.array([3,3]) , (1,2) ) )
```

Model summary and values of the weights

```
model.summary()
```

```
print(model.layers[0].get_weights())
```

```
print(model.layers[1].get_weights())
```

Keras is a high-level neural networks API, written in Python and was developed with a focus on enabling fast experimentation and has extensive documentation.