Technische Universität Berlin

Lecture Notes

Machine Learning II

Grégoire Montavon, summer semester 2020

Contents

List of Figures ..... Page i

1 Low dimensional embedding ..... Page 1

1.1 Recap of PCA 1
1.2 LLE 2
1.3 Multidimensional Scaling (MDS) and IsoMap 9
1.4 Stochastic neighbouring embedding 10
1.5 Analysing non-euclidean pairwise data 13

2 Component Analysis ..... Page 17

2.1 Canonical Correlation Analysis (CCA) 17
2.2 Independent Component Analysis (ICA) 25

3 Kernel Machines ..... Page 33

3.1 Structured Kernels / Inputs 33
3.2 Structured Prediction / Outputs 41
3.3 Kernel for anomaly detection 48

4 Deep Learning ..... Page 54

4.1 Neural Networks for Structured Data 54
4.2 Structured Prediction 63
4.3 Explainable Models 68

5 Federated Learning Page 69

Index Page 72

Lecture notes by Viktor Stein.

These lecture notes are not endorsed by the lecturer or the university and make no claim to accuracy or correctness.

If you find errors please contact v.glombik@gmail.com.

Last edited: April 12, 2022.

List of Figures

1 Three-dimensional data lies of a two-dimensional manifold, a so called Swiss roll. [RS00] . . . 1

2 EMG signal is a collection of highly correlated data; only very few dimensions are needed to explain almost all of the data. . . . 1

3 High dimensional word representation with 1-sparse vectors. . 1

4 Embedded word data representation (in reality, there rather 128 than 3 parameters). . . . 1

5 Data lying close to a two dimensional subspace (plane) is projected onto a plane, where the different classes (encoded by color) are separated well [Sch06]. . . . 2

6 S shaped data is not well reduced by PCA. [PCA] . . . 2

7 LLE projection of the data from above. . . . 2

8 The LLE method [RS00]. . . . 3

9 The top image is the PCA projection, which does not show any structure of the data. The lower image is the LLE projection (with four neighbours), where images with the face at the margin are on the boundary of the two-dimensional manifold and images with the picture in the center are in the center. [RS00] . . . 7

10 LLE can also be applied to text. [RS00] . . . 7

11 A uniform sampling of a swiss role. . . . 8

12 For ten neighbours, the embedding is good. If the noise is too high, even with the "right" number of neighbours or way more, the embedding fails. (The quality of the embedding increases for a higher number of samples.) . . . 8

13 The original data set contains three mutually perpendicular circles in six dimensional space, meeting at one points. The PCA projection (left) does not preserve the structure of the data, the circles are not visible. The Sammon mapping preserves the topological structure: while the circles become distorted, there are still three closed loops meeting at a single point. . . . 9

14 If we measure distances along the manifold, $d(1,6) > d(1,4)$. 9

15 A neighbourhood graph. 10

16 Linear methods cannot interpolate properly between the leftmost and rightmost images in each row, because the middle images are not averages of the outer ones. 10

17 What does this picture mean? 10

18 In contrast to the Gaussian distribution, the student t distribution doesn't decay as fast, it is heavy-tailed. 12

19 [MH08, p. 2587] 13

20 PCA does not work well on MNIST. 13

21 [MH08, p. 2590 - 2591] 13

22 Pairwise data can be represented as undirected graphs, as tables ("matrices") or a checkerboard patterns. 14

23 For conflicting comparisons, one can create separate charts. . 15

24 Projection onto the positive and projection onto the negative eigenspaces yield results different in nature. In the upper left corner, the eigenvalue spectrum are plotted. 15

25 [VdMH12] 16

26 Some variables are highly correlated, but it is impractical to have to look at all 20 correlation plots to learn something about the data. 17

27 todo 18

28 todo 18

29 Bag-of-words feature representation 19

30 TODO! 22

31 First two canonical components of the bag of words kernels for the English (constitution) articles, as obtained by doing CCA with the other languages. Articles from different chapters are represented by a different symbol. 22

32 From left to right: neurophysical signal (?), spectogram of neural activity (frequency dependent HRF (hemodynamic response function, roughly measure blood oxygenation) (?), ?, ?. 22

33 Canonical trend model: We have data X, which is the web source features (bag of words of different web sources, e.g. newspapers, news sites or twitter accounts) and we have geographic features (retweet frequencies at different locations). We want to know the trend, which is a hidden variable, which affects both the content and the retweet frequency. We want to find a mapping from X to Z and from Y to Z. 23

34 **TODO???** 23

35 Performance of Canonical Trends compared to Mean and PCA. 24

36 Performance of Canonical Trends compared to Mean and PCA. 24

37 The blind source separation problem. [Source?] 25

38 **TODO????** 25

39 TODO 26

40 PCA vs ICA. 26

41 BSS of nonlinearly distorted mixtures with kernel based learning methods [HZKM02]. 26

42 The RMSE vs. the uncertainty estimate for the two used algorithms. For small values ($U \leqslant 0.1$) the uncertainty allows to predict the RMSE. [MZKM02] 27

43 [MZKM02] 27

44 TODO [MZKM02] 27

45 [MZKM02] show that JADE separates cardiac signals of mother and fetus in an recording of a pregnant woman: the separability matrix has a block structure, which is of physiological relevance: it indicates independent multi-dimensional subspaces. 27

46 When the feature map is higher dimensional, the expressive power of the model is increased. 33

47 The preprocessing step for the "Bag-of-Words" kernel. 37

48 Biosynthesis. 37

49 The different rows indicate if there are shared n-grams ("#n-mers") between the two sequences x and x'. 37

50 Illustration of just looking for maximum blocks of common subsequences. 37

51 Tree representation of sentences derived from a grammar: Parse tree for "mary at lamb". 38

52 The subtrees of different sizes of the tree considered above. . 38

53 An efficient implementation using dynamic programming visualised. 39

54 In the generator matrix, green is zero, yellow is one and red is a negative number. 40

55 The results show that the higher β, the higher the similarity between points of the same cluster 41

56 Standard machine learning vs structured outputs. 42

57 Geometrical interpretation of the large-margin model. 43

58 Think of this diagram as a Markov chain. Each circle is a state and a column of circles is the set of possible states. Different columns represent the state at different time steps. The path shown with lines depicts a possible sequence of states. Our goal is to find the best possible sequence of states and the variable a represents the cost or reward of transitioning between the different states. 44

59 Geometric interpretation of margin rescaling. 47

60 Geometric interpretation of slack rescaling. 47

61 The lower diagram shows the transition policy between the different states, which our structured outputs model will learn to implement. 47

63 **TODO: Warum ist das nicht ein binary system?** . . . 49

64 One could treat this as a two class problem (with large margin) and find a decision boundary between known anomalies and normal data. This approach might be insufficient, since new anomalies must not obey this decision boundary. We thus must find a way to effectively enclosed the normal data, but it is not clear how to do that. 49

65 **Left:** Geometric interpretation of reconstruction based methods. **Right:** The surface separating the inliers and outliers is adjusted into the direction opposite of the outliers. 49

66 **TODO** 49

67 Consider PCA in $\mathbb{R}^2$ with one principal component, PCA1. For an outlier the norm of the residual is the length of the projection onto PCA1. 50

68 For $a = 1$ (one component is reserved for the PCA model and one for the residual) we get ... for different types of kernels. For the linear model we see that the outlier function increases along the residual components, placing most of the data in the white area. But there are data points, for which the model does not work well. Furthermore there are white regions, which do not contain any inliers. The Gaussian kernel works well locally but not globally, which is clear as $e^{-x} \xrightarrow{x \to \pm\infty} 0$ 50

69 TODO 50

70 **TODO** 52

71 **TODO** 52

72 The explanation highlights which pixels are responsible for the anomaly detection. 53

73 The lines represent kind of level lines, to illustrated the outlier function. 53

74 To build a assistance to drive on likes to detect the lanes and the other vehicles. Based on those features, the classifier might take a decision such as turning left or right. (The vector should rather look like $\ldots, x_3, x_4, \ldots, x_7, x_8, x_9, \ldots$.)54

75 A simple three-layer network 54

76 Constructive "proof" of universality of neural networks. . . . 56

77 Progressive exclusive-or composition. [dai] 56

78 Schematic diagram illustrating the interconnections between layers in the neocognitron. [Fuk80] 57

79 Architecture of a CNN. [NL] 58

80 The convolution layer. 58

81 A convolution operation with zero padding so as to retain in-plane dimensions. [YNDT18] 58

82 TODO 60

83 Based on their similarity, the historical tables can be mapped into an embedded space using e.g. t-SNE. Points that are mapped to the same location, can indeed be verified to have a similar numerical content, they would often correspond to the same table taken from different books. 60

84 A vector embedding on a sentence. 61

85 The "merge" and the "readout" neural network in a parsing tree. 61

86 Applying the readout function on each node allows to visualise how the sentiment builds up in the recursive neural network. [SPW+13] 61

87 The input not layer zero, it is the graph, which is a initial state. At each layer, the GNN preforms at diffusion step in the graph until one achieves a prediction. Thus the input graph is at every layer of the GNN. [SEL+20] 62

88 A Message Passing Neural Network predicts quantum properties of an organic molecule by modelling a computationally expensive DFT calculation. [GSR+17] 62

89 Source? 62

90 [KW16] 62

91 TODO 63

92 **todo: citation** 63

93 Taking the mean of multiple outputs may result in a bad prediction: the mean point is at a point where there are no data points.**todo: citation** 63

94 **todo: citation** 63

95 The output of the network are the parameters $z = (a_i, \mu_i, \sigma_i)$. If we have a mixture of three components, we have three values of a, three vectors μ and three positve scalars σ. **TODO citation 2 from slides** 64

96 [BPT+20] 64

97 One can think of the Boltzmann machine as implementing (linear) decision boundaries in the input space and attributing high probability to points that are on one side and low probability for points on the other side. 64

98 [JyHL+14] 65

99 A model measures the compatibility between observed variables X and variables to be predicted Y using an energy function $E(Y, X)$. For example, X could be the pixels of an image, and Y a discrete label describing the object in the image. Given X, the model produces the answer Y that minimises the energy E. [LCH+06] 66

100 Several applications of energy-based learning: (a) face recognition: Y is a high-cardinality discrete variable: an input image is detected to be Einstein or not Einstein; (b) face detection and pose estimation: Y is compromised of bounding boxes: a collection of vectors with location and pose of each possible face; (c) image segmentation: Y is an image in which each pixel is a discrete label (belonging to the nuclei or not); (d-e) handwriting recognition and sequence labelling: Y is a sequence of symbols from a highly structured but potentially infinite set (the set of English sentences). The situation is similar for many applications in natural language processing and computational biology; (f) image restoration: Y is a high-dimensional continuous variable (an image). [LCH+06] 66

101 For the correct answer Y^i, one wants the energy to be low and for the wrong one (so any other answer that is incorrect) to be high. We receive pairs of outputs, a correct and a incorrect one and one adapts the model W to increase/decrease the energy function accordingly. [LCH+06] 66

102 Examples of loss functions. Margin is an important component to ensure that the learned model generalises well to test data. 67

1 Low dimensional embedding

In applications, one often has reason to believe that high-dimensional data lies close to a lower dimensional subspace, so there are fewer parameters needed to account for the data properties. This is due to hidden causes or latent variables.

Examples for such data are high resolution images, costumer data, neural data or news data.

Example 1.0.1 (Word2Vec embedding)
Word2Vec tries to find a low dimensional embedding of words which is semantically meaningful, so one can detect relations and similarities between words. In the embedding space, the directions from "man" to "woman" and "king" to "queen" coincide. Initially, one represents words as a very high dimensional data, where every dimension is an indicator for a specific word in the alphabet, and each word corresponds to a vector with a one in that position and zero else. One then embeds this data into a continuous space which is more meaningful. ◇

The upsides of dimensionality reduction are

- **Visualisation**: one gains insights into high dimensional structures in the data.
- **Better generalisation:** chance of overfitting is reduced and we gain enhanced representations of the data.
- **Speed** most algorithms scale badly with increasing data dimension (e.g. inverting matrices ($O(n^3)$), computing covariance matrices (in $O(Nn^2)$ for $X \in \mathbb{R}^{N\times n}$ because of the eigendecomposition)).
- **Compression** Lower dimensional data needs less storage, which is especially relevant of IoT (Internet of Things) devices.

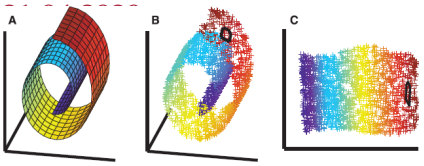


Fig. 1: Three-dimensional data lies of a two-dimensional manifold, a so called Swiss roll. [RS00]

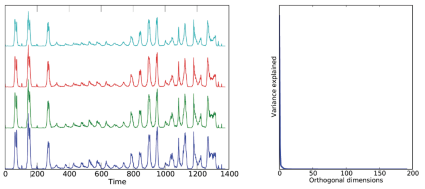


Fig. 2: EMG signal is a collection of highly correlated data; only very few dimensions are needed to explain almost all of the data. Here, LLE is used as a preprocessing step.

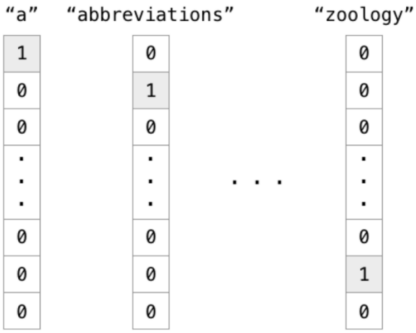


Fig. 3: High dimensional word representation with 1-sparse vectors.

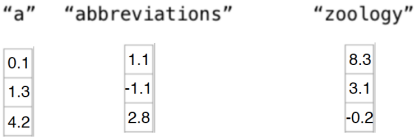


Fig. 4: Embedded word data representation (in reality, there rather 128 than 3 parameters).

1.1 Recap of PCA

For data $X := [x_1, \dots, x_N] \in \mathbb{R}^{D\times N}$, PCA find a direction $w \in \mathbb{R}^D$ such that the variance of the projected data $w^\mathsf{T}X$ is maximal. We have **(TODO: WHAT IS x?? is $\mathbb{E}(x) = \frac{1}{N}\sum_{n=1}^N x_n$??)**

$$\begin{aligned}\mathrm{Var}(w^\mathsf{T}X) &= \frac{1}{N}\sum_{n=1}^N \left(w^\mathsf{T}x_n - \mathbb{E}(w^\mathsf{T}x)\right)^2 = \frac{1}{N}\sum_{n=1}^N \left(w^\mathsf{T}\left(x_n - \mathbb{E}(x)\right)\right)^2 \\ &= \frac{1}{N}\sum_{n=1}^N w^\mathsf{T}\left(x_n - \mathbb{E}(x)\right)\left(x_n - \mathbb{E}(x)\right)^\mathsf{T} w \\ &= w^\mathsf{T}\left(\frac{1}{N}\sum_{n=1}^N \left(x_n - \mathbb{E}(x)\right)\left(x_n - \mathbb{E}(x)\right)^\mathsf{T}\right) w =: w^\mathsf{T}Sw,\end{aligned}$$

by linearity of $\mathbb{E}$, where S is the covariance matrix.

For $Sw = \lambda w$, the maximal variance in direction w is given by

$$\arg\max_w \frac{\mathrm{d}}{\mathrm{d}w} w^\mathsf{T}Sw = \arg\max_w \frac{w^\mathsf{T}Sw}{w^\mathsf{T}w} = \frac{w^\mathsf{T}\lambda w}{w^\mathsf{T}w} = \lambda,$$

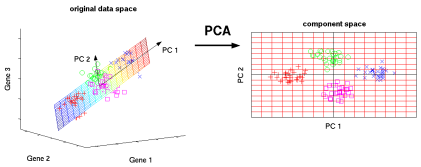
i.e. the variance of the projected data in an eigendirection w is given by the corresponding eigenvalue.

The direction of maximal variance in the data is equal to the eigenvector having the largest eigenvalue. Preforming the eigendecomposition $X = W\Lambda W^\mathsf{T}$ yields $W := [w_1, \dots, w_k] \in \mathbb{R}^{D\times k}$. The projection of a data point x is given by $W^\mathsf{T}x$.

But there are lower dimensional non-linear manifolds for which PCA (a global linear method) fails by not being able to capture the structure of the data, (cf. Fig. 6). Many nonlinear dimension reduction methods exist, such as Kernel PCA, ISOMAP, LLE, Hessian eigenmaps, Diffusion maps, Maximum variance unfolding and many more. To make PCA better adapt to non-linear structures, we implicitly map the data to a higher dimensional (in the case of a GAUSSIAN kernel even infinitedimensional) space and preform standard PCA there ("Kernel trick").

Solving PCA via $X^\mathsf{T}X$ instead of XX^T (note that if X is centered, $\sum_{n=1}^{N} x_n = 0$ and thus $S = \frac{1}{N}XX^\mathsf{T}$) is called linear kernel PCA. The eigendecomposition only depends on the inner products $(XX^\mathsf{T})_{i,k} = \langle x_i, x_k \rangle$. The matrix XX^T can be replaced with a kernel matrix $K_{i,j} := \langle \Phi(x_i), \Phi(x_j) \rangle$, where $\Phi\colon \mathbb{R}^D \to \mathbb{R}^K$ with $K \gg D$ and thus $K \in \mathbb{R}^{K\times K}$. In this higher dimensional space, data can often be linearly separated.

Several nonlinear dimensionality reduction methods can be viewed as (a variant of) kernel PCA with kernels learned form the data [HLMS04], (cf. remark 1.2.7).



eigendecomposition

Fig. 5: Data lying close to a two dimensional subspace (plane) is projected onto a plane, where the different classes (encoded by color) are separated well [Sch06].

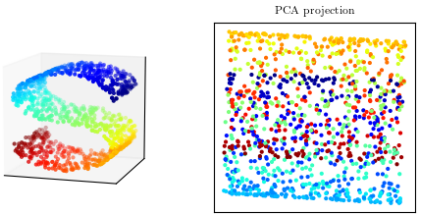


Fig. 6: S shaped data is not well reduced by PCA. [PCA]

linear kernel PCA

1.2 LLE

The idea of LLE is to find a lower dimensional representation of the data that preserves neighbourhood relations. LLE illustrates a general principle of manifold learning, elucidated by Tenenbaum et al., that overlapping local neighbourhoods - collectively analysed - can provide information about global geometry. As more dimensions are added to the embedding space, the existing ones do not change **(???)** . A virtue of LLE is that it avoids the need to solve large dynamic programming problems.

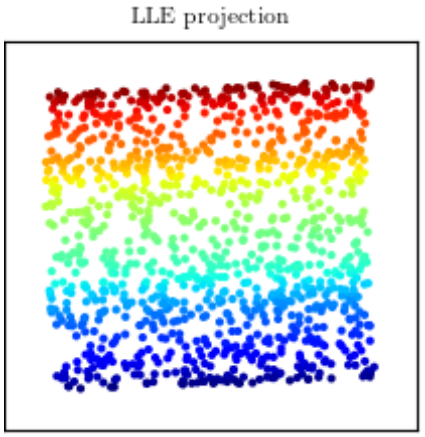


Fig. 7: LLE projection of the data from above.

For each data point X_i select some neighbour points. The number of neighbours (here: eight) used is a meta-parameter of the algorithm. LLE now tries to reconstruct X_i as a linear combination of its neighbours, i.e. it learns weights $W_{i,k}, W_{i,j}, \dots$ such that $X_i \approx W_{i,k}X_k + W_{i,j}X_j + \dots$. Given these weights, LLE tries to find a lower dimensional embedding Y_i which can be reconstructed by its neighbours $Y_k, Y_j, \dots$, i.e. $Y_i \approx W_{i,k}Y_k + W_{i,j}Y_j + \dots$.

In summary, LLE extracts a local feature (fit) and then makes sure that it is preserved in the lower dimensional projection. This is a common technique for dimensionality reduction.

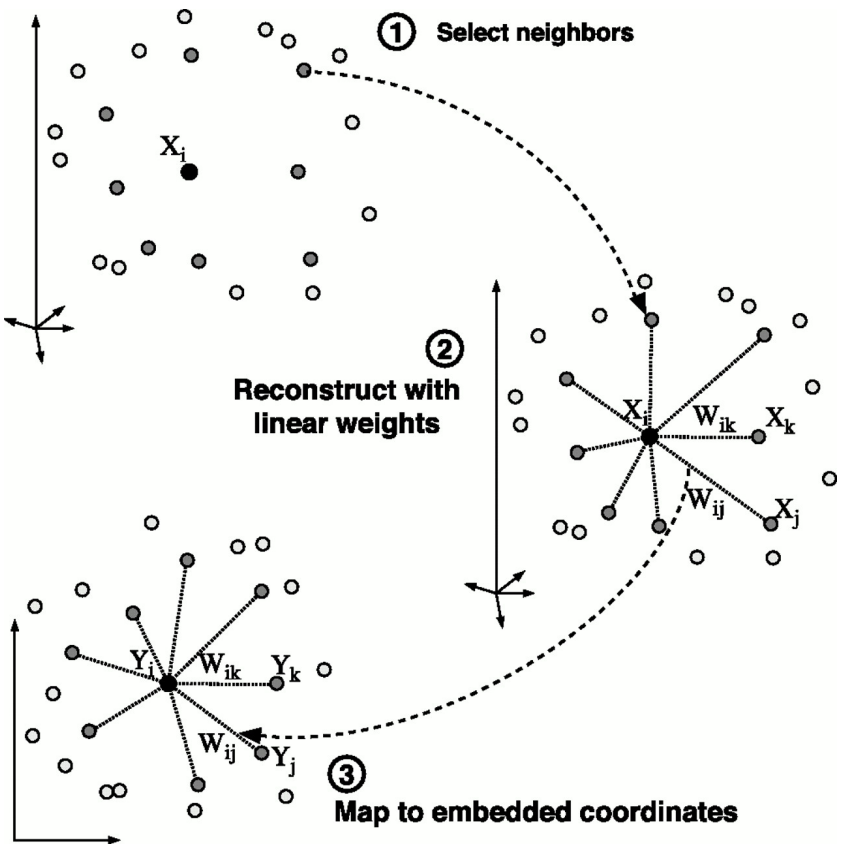


Fig. 8: The LLE method [RS00].

We can write the LLE algorithm in three steps.

1. For of each data point x_i, $i \in \{1, \dots, n\}$, compute its neighbours $(x_j)_{j \in N_i}$.
2. Compute the weights w_{ij} that be reconstruct x_i from its neighbours, minimising the squared loss

$$R\left(W := [(w_{1j})_{j \in N_1} \dots (w_{nj})_{j \in N_n}]\right) := \sum_{i=1}^{n} \left\| x_i - \sum_{j \in N_i} w_{ij} x_j \right\|^2 \quad (1)$$

$$\text{subject to } \sum_{j \in N_i} w_{ij} = 1 \ \forall i \in \{1, \dots, n\}. \quad (2)$$

   by constraint linear fits.
3. Compute the (embedded) vectors y_i, $i \in \{1, \dots, n\}$ best reconstructed by the weights w_{ij}, minimising the quadratic form

$$\Phi(Y := [y_1, \dots, y_n]) := \sum_{i=1}^{n} \left\| y_i - \sum_{j \in N_i} w_{ij} y_j \right\|^2$$

Remark 1.2.1 (Symmetries in the LLE objective function)
For any data point, the minima of $R(W)$ are invariant to rotations, rescalings and translations of the data point and its neighbours. The reconstruction weights thus characterise intrinsic geometric properties of each neighbourhood independent of the frame of reference. ◊

Proof. (HA 1-1(a)) (1) Replacing all x_i by ax_i for $a > 0$, we have

$$\sum_{i=1}^{n} \left\| \alpha x_i - \sum_{j \in N_i} w_{ij} \alpha x_j \right\|^2 = \alpha^2 \sum_{i=1}^{n} \left\| x_i - \sum_{j \in N_i} w_{ij} x_j \right\|^2 .$$

Thus $R(W)$ and its transformed version only differ by a constant. Hence their minimisers coincide.

(2) Replacing all x_i by $x_i + v$ for $v \in \mathbb{R}^n$, we have

$$\begin{aligned} \left\| x_i + v - \sum_{j \in N_i} w_{ij}(x_j + v) \right\|^2 &= \left\| x_i + v - \sum_{j \in N_i} w_{ij} x_j - v \underbrace{\sum_{j} w_{ij}}_{=1 \ \forall i} \right\|^2 \\ &= \left\| x_i - \sum_{j \in N_i} w_{ij} x_j \right\|^2 , \end{aligned}$$

highlighting the importance of (2).

(3) As the euclidean norm is invariant under orthogonal transformations, replacing all x_i by Ux_i, where $U \in \mathbb{R}^{n \times n}$ is an orthogonal matrix, yields

$$\begin{aligned} \sum_{i=1}^{n} \left\| Ux_i - \sum_{j \in N_i} w_{ij} U x_j \right\|^2 &= \sum_{i=1}^{n} \left\| U \left(x_i - \sum_{j \in N_i} w_{ij} x_j \right) \right\|^2 \\ &= \sum_{i=1}^{n} \left\| x_i - \sum_{j \in N_i} w_{ij} x_j \right\|^2 . \end{aligned}$$

□

Remark 1.2.2 (Step 2 is local, step 3 is global)
The reconstruction weights w_{ij} for each data point x_i are computed from its local neighbourhood and are thus independent of the weights for other data points. However, the embedding coordinates y_i are computed by an $n \times n$ eigensolver, coupling all data points. This is how the algorithm leverages overlapping local information to discover global structure. ◊

Step 2 of LLE

We can find the minima of $R(w)$ by using LAGRANGE multipliers. As the summands are independent of each other (cf. the remark above), the objective function can be decomposed as a sum of as many subobjectives as there are data points: we can rewrite (1) as

$$R(W) = \sum_{i=1}^{n} R_i(x_i, w_i),$$

i.e. $R_i(x_i, w_i) := \left\| x_i - \sum_{j \in N_i} w_{ij} x_j \right\|^2$, where $w_i := (w_j)_{j \in N_i}$.

(HA 1-1 (b)) As we only need to look at one summand, we can, for convinience of notation assume that $x := x_i$ has K neighbours:

$$R_i(x_i, w_i) := R(w) := \left\| x - \sum_{j=1}^{K} w_j \eta_j \right\|^2 ,$$

where $\eta := (\eta_1, \ldots, \eta_K)^\mathsf{T} \in \mathbb{R}^{K \times n}$ contains the K nearest neighbours of x and $w := (w_j)_{j=1}^{K} := (w_{i,j})_{j \in N_i}$ is subject to $w^\mathsf{T} \mathbb{1} = 1$.

We then have

$$\left\| x - \sum_{j=1}^{K} w_j \eta_j \right\|^2 = \left\| x\,\mathbb{1}^\mathsf{T} w - \eta^\mathsf{T} w \right\|^2 = \left\| (x\,\mathbb{1}^\mathsf{T} - \eta^\mathsf{T}) w \right\|^2$$
$$= w^\mathsf{T} (\mathbb{1}\,x^\mathsf{T} - \eta)(x\,\mathbb{1}^\mathsf{T} - \eta^\mathsf{T}) w = w^\mathsf{T} C w$$

for $C \coloneqq \left(\mathbb{1}\,x^\mathsf{T} - \eta\right)^\mathsf{T} \left(\mathbb{1}\,x^\mathsf{T} - \eta\right)$. To find an analytic form of the minimum w we define the LAGRANGIAN

$$\mathcal{L}(w, \lambda) \coloneqq w^\mathsf{T} C w - \lambda(w^\mathsf{T}\,\mathbb{1} - 1),$$

whose partial derivatives are (as C is symmetric)

$$\frac{\partial \mathcal{L}(w,\lambda)}{\partial w} = 2Cw - \lambda\,\mathbb{1} \quad \text{and} \quad \frac{\partial \mathcal{L}(w,\lambda)}{\partial \lambda} = 1 - w^\mathsf{T}\,\mathbb{1}\,. \tag{3}$$

Setting the first equation to zero leads to $2wC = \lambda\,\mathbb{1}$ and thus $w = \frac{1}{2}\lambda C^{-1}\,\mathbb{1}$. Hence we have

$$1 \overset{!}{=} w^\mathsf{T}\,\mathbb{1} = \frac{\lambda}{2}(C^{-1}\,\mathbb{1})^\mathsf{T}\,\mathbb{1} = \frac{\lambda}{2}\,\mathbb{1}^\mathsf{T}\,C^{-\mathsf{t}}\,\mathbb{1} = \frac{\lambda}{2}\,\mathbb{1}^\mathsf{T}\,C^{-1}\,\mathbb{1},$$

which yields

$$\lambda = \frac{2}{\mathbb{1}^\mathsf{T}\,C^{-1}\,\mathbb{1}},$$

implying

$$w = \frac{\lambda}{2} C^{-1}\,\mathbb{1} = \frac{C^{-1}\,\mathbb{1}}{\mathbb{1}^\mathsf{T}\,C^{-1}\,\mathbb{1}}.$$

As C is positive definite and the linear constraints are convex, this is indeed a minimiser.

Remark 1.2.3 If the covariance matrix C is (nearly) singular, one can instead consider $C + \frac{\Delta}{K} I_K$, where we choose $0 < \Delta \ll \mathrm{Tr}(C)$. ◊

Remark 1.2.4 (HA 1-1(c)) The optimal w can also be found by solving $Cw = \mathbb{1}$ and then rescaling w such that $w^\mathsf{T} 1 = 1$, which can be computationally faster: If $w = C^{-1}\,\mathbb{1}$, then $1 = w^\mathsf{T}\,\mathbb{1} = (C^{-1}\,\mathbb{1})^\mathsf{T}\,\mathbb{1} = 1^\mathsf{T} C^{-T}\,\mathbb{1}$ and thus $w = \frac{C^{-1} w}{1^\mathsf{T} C^{-T}\,\mathbb{1}}$. ◊

Step 3 of LLE

We want to find the matrix $Y \coloneqq [y_1, \ldots, y_n]^\mathsf{T} \in \mathbb{R}^{n \times p}$ which minimises $\Phi(Y)$. Let $w_i \coloneqq (w_{ij})_{j=1}^{n}{}^\mathsf{T}$, where $w_{ij} = 0$ if $j \notin N_i$, as we adapt to global optimisation here, cf. the remark 1.2.2. We can then write

$$\Phi(Y) = \sum_{i=1}^{n} \left\| y_i - Y^\mathsf{T} w_i \right\|^2 = \sum_{i=1}^{n} \left(y_i^\mathsf{T} y_i - w_i^\mathsf{T} Y y_i - y_i^\mathsf{T} Y^\mathsf{T} + w_i^\mathsf{T} Y Y^\mathsf{T} w_i \right).$$

Each summand is a quadratic form $\sum_{i=1}^{n} a_i^\mathsf{T} M b_i$. Recall that for $A = [a_1, \ldots, a_n]^\mathsf{T}$ and B similarly we have $\sum_{i=1}^{n} a_i^\mathsf{T} M b_i = \mathrm{Tr}[AMB^\mathsf{T}]$.

Letting $W \coloneqq [w_1, \ldots, w_n]^\mathsf{T}$ this implies

$$\Phi(Y) = \mathrm{Tr}\left(YY^\mathsf{T} - WYY^\mathsf{T} - YY^\mathsf{T}W^\mathsf{T} + WYY^\mathsf{T}W^\mathsf{T}\right)$$
$$= \mathrm{Tr}\left((Y - WY)(Y^\mathsf{T} - Y^\mathsf{T}W^\mathsf{T})\right) = \mathrm{Tr}\left((I_n - W)YY^\mathsf{T}(I_n - W)^\mathsf{T}\right),$$

as the trace is invariant under cyclic permutations. We conclude

$$\Phi(Y) = \operatorname{Tr}\left(Y^{\mathsf{T}}(I_n - W)^{\mathsf{T}}(I_n - W)Y\right) =: \operatorname{Tr}\left(Y^{\mathsf{T}}MY\right) = \sum_{k=1}^{p} Y_k^{\mathsf{T}} M Y_k,$$

where $M := (I_n - W)^{\mathsf{T}}(I_n - W)$ Y_k is the k-th column of Y.

We can assume that Y_k form an orthonormal basis, i.e $Y_i^{\mathsf{T}} Y_j = \delta_{ij}$. **WHYYYY?**

We have $\mathbb{1}_n^{\mathsf{T}} Y_j = 0$: as $w^{\mathsf{T}}\mathbb{1} = 1$ we have $(I_n - W)\mathbf{1} = \mathbb{1} - \mathbb{1} = 0$. Therefore we can subtract any number from the vector Y_j. If we choose this number a to be the mean of Y_j, we obtain

$$(I_n - W)(Y_j - a\,\mathbb{1}) = (I_n - W)Y_j - 0.$$

Thus the solution consists of the p eigenvectors om M orthogonal to $\mathbb{1}$ corresponding to the smallest eigenvalues. **WHYYYY?** The columns of Y are these eigenvectors and the new locations (in p dimensions) are the corresponding rows $y_1^{\mathsf{T}}, \ldots, y_n^{\mathsf{T}}$ of Y.

Remark 1.2.5 (Complexity of LLE)
The complexity of step 1 is $O(dn^2)$ (but can be reduced to $O(n\log(n))$ with kd-trees), the complexity of step 2 is $O(dnk^3)$ and of step 3 is $O(dn^2)$ (but methods from sparse eigenproblems can reduce it further).◊

Remark 1.2.6 (LLE from pairwise distances)
LLE can be applied to user input in the form of pairwise distances. In this case, nearest neighbours are identified by the smallest nonzero elements of each row of the distance matrix.

To derive the reconstruction weights for each data point, we need to compute the local covariance matrix C between its nearest neighbours given by

$$c_{jk} = \frac{D_j + D_k - D_{jk} - D_0}{2},$$

where D_{jk} denotes the squared distance between the j-th and k-th neighbours $D_j := \sum_z D_{jz}$ and $D_0 := \sum_{jk} D_{jk}$. **(SO** $c_{jk} \leqslant 0$**???)** ◊

Remark 1.2.7 (Kernel view of LLE) Coordinates of the eigenvectors $2, \ldots, p+1$ provide the LLE embedding [HLMS04].

LLE can be viewed as a certain instance of kernel PCA, with a kernel $K := \lambda_{\max} I - M$, where $M := (I - W)(I - W)^{\mathsf{T}}$ is learned from the data: the i-th row of W contains the linear coefficients that sum to unity and optimally reconstruct x_i from its p nearest neighbours. ◊

Remark 1.2.8 (Limitations of LLE)
LLE

- is sensitive to noise
- is sensitive to non-uniform sampling of the manifold (common limitation).
- does not provide an explicit mapping, though one can be learned in a supervised fashion from the pairs (x_i, y_i).
- has quadratic complexity on the training size (cf. remark 1.2.5).

- isn't a robust method to compute the intrinsic dimensionality (unlike ISOMAP).
- doesn't have robust method to define the neighbourhood size K (meta parameter). If K is chosen badly, the embedding suffers. ◊

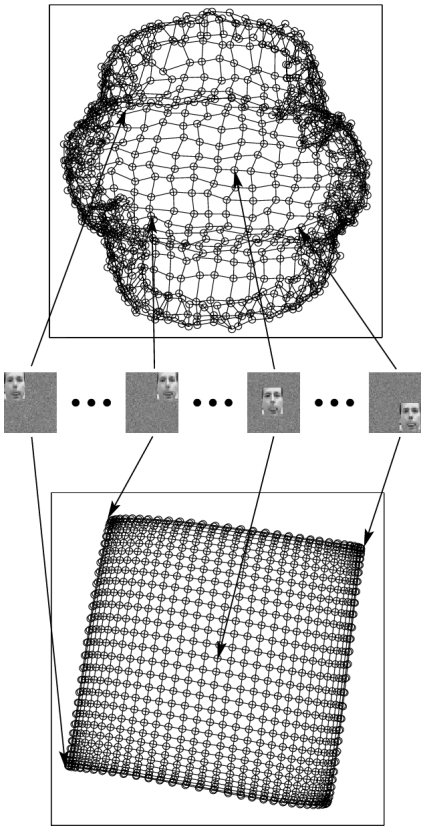


Fig. 9: The top image is the PCA projection, which does not show any structure of the data. The lower image is the LLE projection (with four neighbours), where images with the face at the margin are on the boundary of the two-dimensional manifold and images with the picture in the center are in the center. [RS00]

Example 1.2.9 (Applications of LLE)
The 961 images all consists of a picture of face, which is translated over a noisy background. Such images lie on an intrinsically two dimensional manifold but have an extrinsic dimensionality equal to the number of pixels in each image (3009). Here different lip expressions are embedded into a two dimensional manifold with LLE (24 neighbours). It is meaningful, as one can clearly see a path of changing expressions.

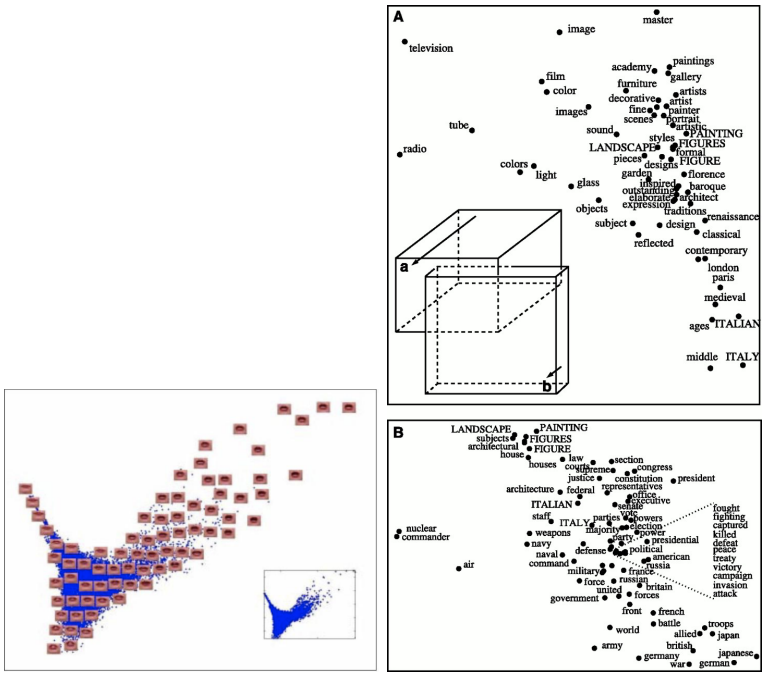


Fig. 10: LLE can also be applied to text. [RS00]

```python
import numpy as np
from matplotlib import pyplot as plt
import sklearn,sklearn.datasets

import numpy.linalg

def LLE(X,k):
    N = len(X)
    W = np.zeros([N,N])

    for i in range(N):
        x = X[i]                                    # extract current data point

        # step 1
        dist = ((x - X)**2).sum(axis = 1)**.5       # euclidean distance, dist.shape = (1000, )
        dist[i] = float('inf')                      # a point is not its own neighbour
```

```
        ind = dist < sorted(dist)[k]                    # boolean array

        # step 2
        diff = x - X[ind]
        C = np.dot(diff, diff.T)                        # local covariance matrix

        # step 3 (see HA 1-2 (c))
        C_stable = C + 0.05*np.identity(k)              # to guarantee that C is invertible
        w = np.linalg.solve(C_stable, np.ones(k))
        w = w / w.sum()                                 # rescale w such that w.sum() = 1
        W[i,ind] = w                                    # fill weight matrix W

    M = np.identity(N) - W - W.T + np.dot(W.T,W)        # M = (I_N - W)' (I_n - W)
    E = np.linalg.svd(M)[0][:,-3:-1]

    return E

X,T = sklearn.datasets.make_swiss_roll(n_samples=1000, noise=0.25) # X.shape = (1000, 3), T.shape = (1000, ).
plt.figure(figsize=(10,10))
ax = plt.gca(projection='3d')
ax.view_init(elev=10., azim=105)
ax.scatter(X[:,0],X[:,1],X[:,2],c=T)
plt.show()

f = plt.figure(figsize=(12,3))
for t,(k,noise) in enumerate([(2,0.1),(10,0.1),(25,0.1),(10,1)]):
    X,T = sklearn.datasets.make_swiss_roll(n_samples=1000, noise=noise)
    embedding = LLE(X,k=k)
    ax = f.add_subplot(1,4,t+1)
    ax.set_title('$k$=%d, noise=%.1f'%(k,noise))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.scatter(embedding[:,0],embedding[:,1],c=T)
```

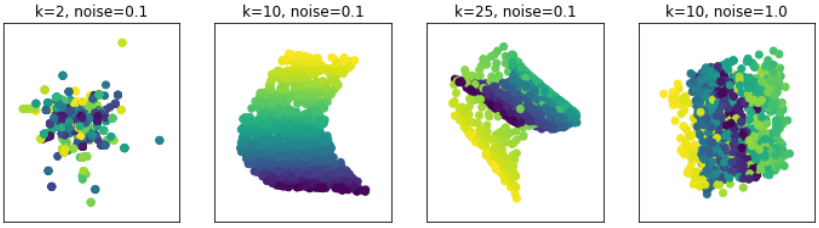


Fig. 12: For ten neighbours, the embedding is good. If the noise is too high, even with the "right" number of neighbours or way more, the embedding fails. (The quality of the embedding increases for a higher number of samples.)

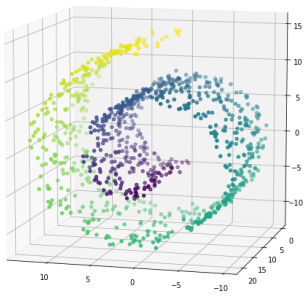


Fig. 11: A uniform sampling of a swiss role.

1.3 Multidimensional Scaling (MDS) and IsoMap

We have seen that PCA is a popular (global linear) method of dimensionality reduction, which finds the directions that have the most variance in the data set by representing where each data point is located along theses axes, in order to minimise the reconstruction error.

A similar, related concept is MDS, which arranges the low-dimensional data points so as to minimise the discrepancy between the pairwise distance in the original space and the pairwise distances in the lower dimensional space (i.e. preserving distances in the lower dimensional space).

A variant of MDS is Metric MDS (MMDS), which minimises

Metric MDS

$$\sum_{i<j}(\|x_i - x_j\|_2^2 - \|y_i - y_j\|_2^2)^2, \tag{4}$$

where x_i and x_j are data points in the original space and y_i and y_j the embedded points, by gradient descent.

This looks similar to PCA, where the projections minimise the squared error. Furthermore, one can show that the "subspace reconstruction error" **(ODER SO)** is minimised. If the data points lie on a hyperplane, their pairwise distances are perfectly preserved by projecting the high-dimensional coordinates onto the hyperplane. In this particular case, PCA is the correct solution. If we double-center the data (row *and* column means are zero) MMDS is equivalent to PCA **(WHYYY?)**, so we don't have to rely on iterative gradient descent but can explicitly find the solution.

The following non-linear extension of the simple cost function (4), which puts more importance on the small distances, is used in a MDS method called Sammon mapping:

$$\sum_{ij}\left(\frac{\|x_i - x_j\| - \|y_i - y_j\|}{\|x_i - x_j\|}\right)^2.$$

It puts too much emphasis on getting very small distances exactly right and thus produces embeddings that are circular with roughly uniform density of the map points.

IsoMap is a popular technique which, instead of modelling local distances, measures the distances across the manifold and then models these intrinsic distances. The main problem is to find a robust way of measuring distances along the manifold. If we have all true distances, then we can apply global MDS (i.e PCA) and we are done. IsoMap provides us with a tool to construct these distances. In contrast, LLE only considers local neighbourhoods, which is not appropriate for the data set in Fig. 14.

IsoMap's method is similar; it connects each data point to its K nearest neighbours in the original space. Assuming the manifold is locally euclidean, it puts the true euclidean distance on each of these links. It

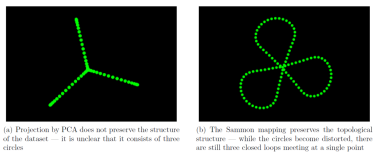


Fig. 13: The original data set contains three mutually perpendicular circles in six dimensional space, meeting at one points. The PCA projection (left) does not preserve the structure of the data, the circles are not visible. The Sammon mapping preserves the topological structure: while the circles become distorted, there are still three closed loops meeting at a single point.

Sammon mapping

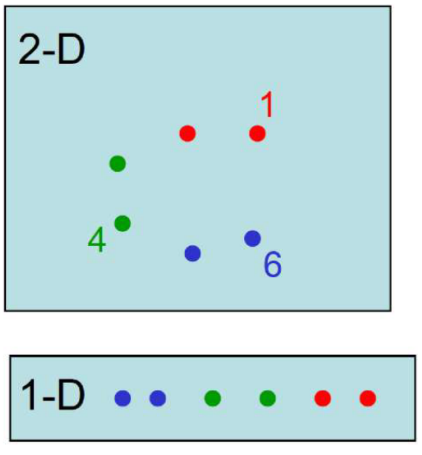


Fig. 14: If we measure distances along the manifold, $d(1,6) > d(1,4)$.

then approximates the manifold distance between any pair of points as the shortest path in this "neighbourhood graph".

The IsoMap approach allows to interpolate between the left- and rightmost images in each row. If one would only take euclidean distance (such as between 1 and 6 in the above figure) this would not work, because this corresponds to averaging the left and rightmost picture, not taking into account the manifold structure.

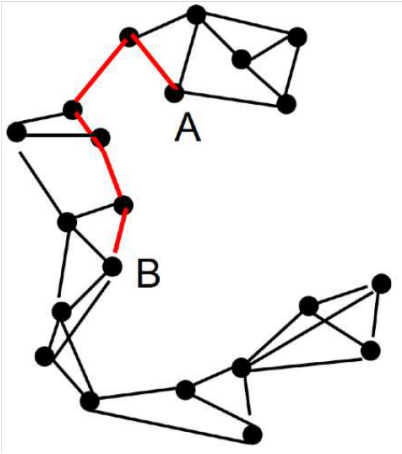


Fig. 15: A neighbourhood graph.

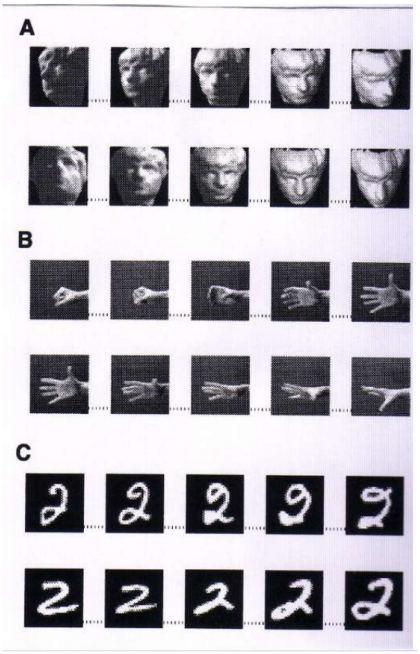


Fig. 16: Linear methods cannot interpolate properly between the leftmost and rightmost images in each row, because the middle images are not averages of the outer ones.

IsoMap does not interpolate properly either because it can only use examples from the training set. It cannot create new images, but it is better than linear methods: when the training set is dense enough, the interpolations look good.

1.4 Stochastic neighbouring embedding

This is a stochastic method, which represents the similarities between neighbours as a (e.g. Gaussian) distribution:

$$p_{ij} := \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}{\sum_{k \neq \ell} \exp\left(-\frac{\|x_k - x_\ell\|^2}{2\sigma^2}\right)}.$$

The density will be larger when the points are closer than when they are far away. The idea is that one selects neighbours stochastically; starting from an x_i one selects a neighbours x_j by using the density. This is called stochastic neighbour selection.

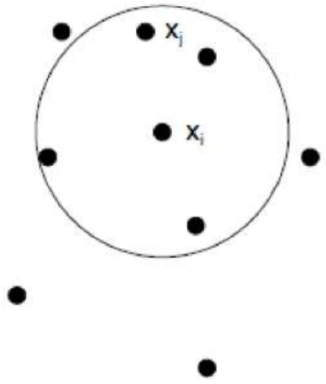


Fig. 17: What does this picture mean?

stochastic neighbour selection

We can do the same in the embedded space:

$$q_{ij} \coloneqq \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq\ell}\exp\left(-\|y_k - y_\ell\|^2\right)}.$$

Here we can intrinsically select the variance in the embedded space, whereas we can impose it in the original space. The objective to minimise is the KL divergence:

$$C \coloneqq KL(P\|Q) = \sum_{ij} p_{ij}\log\left(\frac{p_{ij}}{q_{ij}}\right),$$

which is a measure of matching between two distributions, subject to

$$\sum_{i,j=1}^{N} p_{ij} = \sum_{i,j=1}^{N} q_{ij} = 1.$$

We want to match the distribution in the original space and the one created in the mapped space.

Specifically, the algorithm minimises q, which itself is a function of the coordinates in the embedded space, by gradient descent. We have

$$\frac{\partial C}{\partial q_{ij}} = \frac{\partial}{\partial q_{ij}}\sum_{i,j=1}^{N}\left(p_{ij}\log\left(p_{ij}\right) - p_{ij}\log(q_{ij})\right) = \frac{\partial}{\partial q_{ij}} -p_{ij}\log(q_{ij}) = -\frac{p_{ij}}{q_{ij}}.$$

When the probability matrix q is reparametrised with the softargmax function, i.e.

$$q_{ij} = \frac{e^{z_{ij}}}{\sum_{k,\ell=1}^{N} e^{z_{k\ell}}},$$

the variables z_{ij} can be interpreted as unnormalised log-probabilities. We have

$$\begin{aligned}
\frac{\partial C}{\partial z_{ij}} &= \frac{\partial}{\partial z_{ij}}\sum_{i,j=1}^{N}\left(p_{ij}\log\left(p_{ij}\right) - p_{ij}\left(z_{ij} - \log\left(\sum_{k,\ell=1}^{N} e^{z_{k\ell}}\right)\right)\right) \\
&= \frac{\partial}{\partial z_{ij}}\sum_{i,j=1}^{N} -p_{ij}z_{ij} - \underbrace{\left(\sum_{i,j=1}^{N} p_{ij}\right)}_{=1}\log\left(\sum_{k,\ell=1}^{N} e^{z_{k\ell}}\right) \\
&= -p_{ij} + \frac{\partial}{\partial z_{ij}}\log\left(\sum_{k,\ell=1}^{N} e^{z_{k\ell}}\right) = \frac{e^{z_{ij}}}{\sum_{k,\ell=1}^{N} e^{z_{k\ell}}} - p_{ij} = q_{ij} - p_{ij}.
\end{aligned}$$

In gradient descent, the update will be

$$q_{ij} \leftarrow q_{ij} - \gamma\frac{\partial C}{\partial q_{ij}} = q_{ij} + \gamma\frac{p_{ij}}{q_{ij}} \quad \text{or} \quad z_{ij} \leftarrow z_{ij} - \gamma\frac{\partial C}{\partial z_{ij}} = z_{ij} - \gamma(q_{ij} - p_{ij})$$

for some meta parameter γ. For the first variant we notice that if $q_{ij} \to 0$, the gradient is unbounded and unstable, whereas the second choice is bounded and stable.

In order for to maintain a valid probability distribution, we require $q_{ij} \geqslant 0$ and $\sum_{i,j=1}^{N} q_{ij} = 1$. These properties might not be satisfied after some updates, whereas the z_{ij} can be any real numbers, so no constraints can be violated.

If the scores z_{ij} are reparametrised as $z_{ij} := -\|y_i - y_j\|^2$, where the coordinates $y_i, y_j \in \mathbb{R}^h$ of the embedded data points appear explicitly, we have (as $z_{ij} = z_{ji}$) by the chain rule

$$\frac{\partial C}{\partial y_i} = \sum_{j=1}^{N} \frac{\partial C}{z_{ij}} \frac{\partial z_{ij}}{\partial y_i} + \frac{\partial C}{z_{ji}} \frac{\partial z_{ji}}{\partial y_i} = 2 \sum_{j=1}^{N} \frac{\partial C}{z_{ij}} \frac{\partial z_{i,j}}{\partial y_i}$$
$$= 2 \sum_{j=1} (-p_{ij} + q_{ij})(-2)(y_i - y_j) = 4 \sum_{j=1}^{N} (p_{ij} - q_{ij})(y_i - y_j).$$

In the paper they instead define

$$p_{j|i} := \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_k - x_i\|^2}{2\sigma^2}\right)},$$

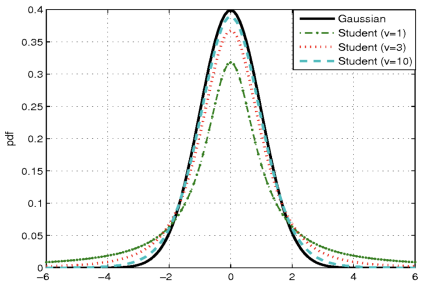
$q_{j|i}$ similarly and thus work with

$$C = \sum_i KL(P_i \| Q_i) = \sum_{ij} p_{j|i} \log\left(\frac{p_{j|i}}{q_{j|i}}\right), \tag{5}$$
$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$
$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\partial C}{\partial Y} + \alpha(t)\left(Y^{(t-1)} - Y^{(t-2)}\right),$$

where η is the learning rate and α the momentum (??) at iteration t. Physically, the gradient may be interpreted as the resultant force created by a set of springs between the map point y_i and all other map points y_j. All springs exert a force along the direction $y_i - y_j$. The spring between y_i and y_j repels or attracts the map points depending on whether the distance between the two in the map is too small or too large to represent the similarities between the two high-dimensional data points. The force exerted by the spring between y_i and y_j is proportional to its length and also proportional to its stiffness, which is the mismatch $p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}$ between the pairwise similarities of the data points and the map points [MH08].

Remark 1.4.1 (Crowding problem) This approach has a fundamental problem, which stems from the fact that we map points from a higher dimensional space to a lower dimensional space. Suppose we want to map our data from a ten dimensional space to a two dimensional space. By preserving this local structure, the dissimilar points, which are not close but not far away, have to be modelled as too far away apart in the map. This is because the Gaussian distribution decays very quickly. Using SNE with the student's t-distribution is called t-SNE.

In a ten dimensional space, one can fit eleven points equidistantly, which is not possible in a two dimensional space. In the words of van der Maaten: the area of the two dimensional map that is available to accommodate moderately distant data points will not be nearly large enough compared with the area available to accommodate nearby data points. Hence, if we want to model the small distances accurately in the map, most of the points that are at a moderate distance from data point i will have to be placed much too far away in the two-dimensional map. [MH08, sec 3.2, p. 2854-2855] ◊



t-SNE

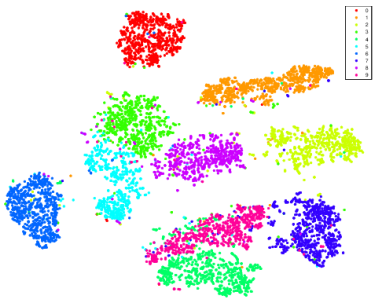
Fig. 18: In contrast to the Gaussian distribution, the student t distribution doesn't decay as fast, it is heavy-tailed. Thus instead using $q_{ij} := \frac{(1+\|y_i - y_j\|^2)^{-1}}{\sum_{k \neq \ell}(1+\|y_k - y_\ell\|^2)^{-1}}$ eliminates the crowding problem, as dissimilar objected are allowed to be modelled too far apart.

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

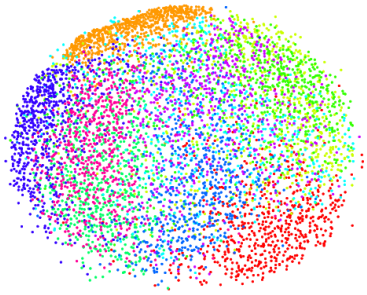
Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=1 **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

Fig. 19: [MH08, p. 2587]

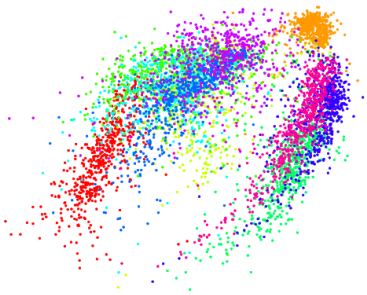
Example 1.4.2 (t-SNE on MNIST)
The MNIST data set consists of 70000 images of handwritten digits (ergo ten classes). LLE also does not preform well, whilst t-SNE works well.



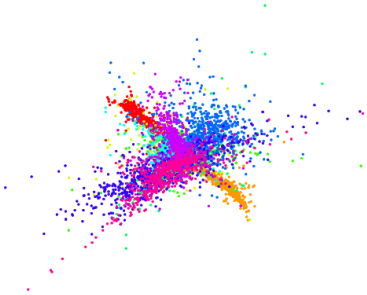
(a) Visualization by t-SNE.



(b) Visualization by Sammon mapping.



(c) Visualization by Isomap.



(d) Visualization by LLE.

Fig. 21: [MH08, p. 2590 - 2591]

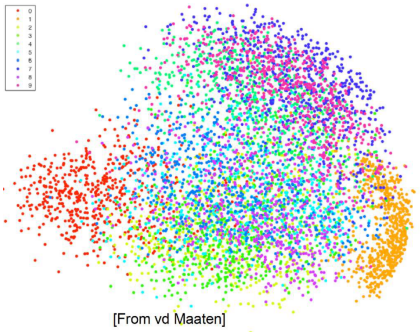


Fig. 20: PCA does not work well on MNIST.

1.5 Analysing non-euclidean pairwise data

Often, the euclidean approach taken in the previous subsection is not possible. In many areas, pairwise data occur and one can compute (dis)similarities but the setting might be non-euclidean or even non-metric. Examples are genomics, text mining, cognitive psychology, social sciences and many more. When you ask people to rate similarity, these

ratings might not be transitive: If x is more similar to y and y is more similar to z, not necessarily, x is similar to z.

In some cases, we have a metric on the dissimilarity matrix D. If it is euclidean, we are lucky. If not, we can still apply MDS, but sometimes the data is even non-metric. It then cannot be represented isometrically as vectors, even in high dimensions.

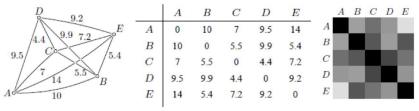


Fig. 22: Pairwise data can be represented as undirected graphs, as tables ("matrices") or a checkerboard patterns.

Definition 1.5.1 (Dissimilarity matrix)
A dissimilarity matrix $D = (d_{i,j})$ is metric if $d_{ij} \geqslant 0$ and $d_{ij} = d_{ji}$ hold for all i, j, $d_{ii} = 0$ holds for all i and $d_{ij} \leqslant d_{ik} + d_{jk}$ holds for all i, j, k.

The matrix D is squared Euclidean if there exists vectors $x_1, \dots, x_n \in \mathbb{R}^p$ such that $d_{ij} = \|x_i - x_j\|_2^2$.

Metric violations translate into indefinite pseudo-covariance matrices. We get problems with applying our standard algorithms here. From D compute $C := -\frac{1}{2}QDQ$, which has p positive and q negative eigenvalues. We kind of double the dimensionality and have a positive and negative part and thus extend the space. In the one part, the vector products are positive and in the other they are negative (can be thought of like extending real numbers by imaginary numbers: for $a \in \mathbb{R}$, $a^2 \geqslant 0$, which is not the case for $a \in \mathbb{C} \setminus \mathbb{R}$).

If the metric properties are not fulfilled, one can incorporate the data into this constructed pseudo-metric space.

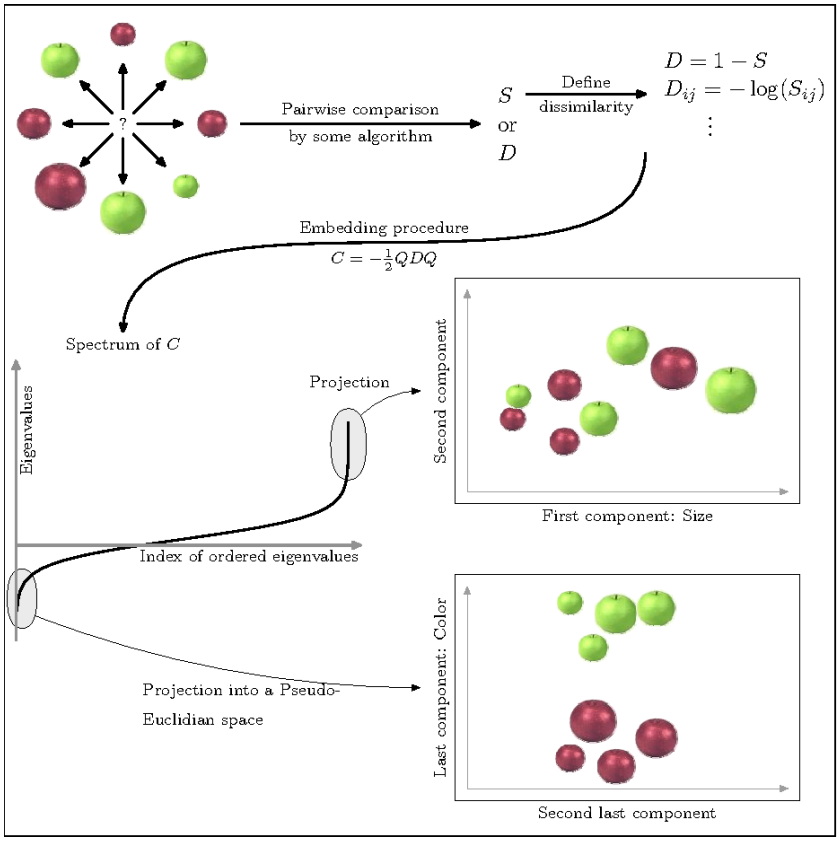


Fig. 23: The data consists of apples with different color and size. Assume we have some pairwise comparisons (e.g. ratings or similarities). If those similarities are conflicting (there is not just one trend; e.g. people generally prefer large over small apples, but for red apples, it is different), so one cannot map into one metric space. To achieve an embedding on this data, one can create two separate spaces, where in one we have the positive parts / eigenvalues, i.e. the scalar product is positive (see graph in the center left). In the first space, the size is the clear component, whereas in the second one, it is color. [LM04]

Example 1.5.2 (Handwritten digits)
The data consists of handwritten digits zero and seven, which either have a bold or a thin stroke. The similarity matrix is obtained from binary image matching ($s_{rs} = \frac{a}{\min(a+b,a+c)}$) on the digits zero and seven.

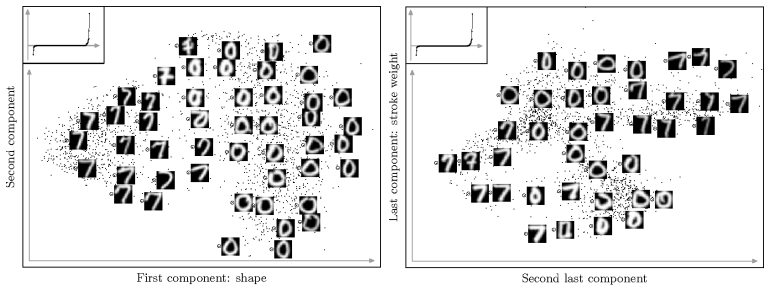


Fig. 24: Projection onto the positive and projection onto the negative eigenspaces yield results different in nature. In the upper left corner, the eigenvalue spectrum are plotted. [LM04]

We are also interested in using this technique for text, where one can construct similarity measures, where metric violations occur also. By extending these spaces to pseudo-euclidean spaces, one can deal with conflicting non-metric similarity ratings.

Remark 1.5.3 (The current paradigm is incomplete [LM04])
Metric violations *can* carry relevant information but a complete data exploratory research must specifically study this information. ◊

Remark 1.5.4 (Limitations of metric spaces)
Metric spaces cannot model intransitive similarities nor objects with high centrality nor asymmetric similarities. Lead Tversky (among others) thus rejected MDS as a model for semantic representation. ◊

Example 1.5.5 (NIPS authors)
When modelling NIPS (a journal) authors (removing co-authors and authors with only one paper), $p_{j|i}$ is the probability that j is the author of a paper of which i is an author. These similarities are likely to be intransitive and asymmetric. ◇

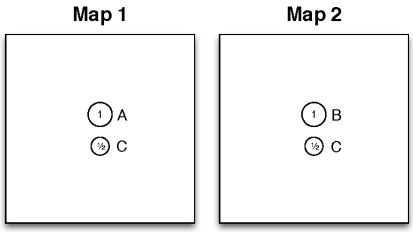
Example 1.5.6 (Multiple Maps t-SNE)
Multiple Maps t-SNE circumvents these limitations by construction multiple maps instead of a single one, where each object has a point and a weight in all maps such that the weights across all maps sum up to one for each object.

As input, it takes the $(p_{j|i})_{i,j}$ and the similarity between i and j under the model is given by

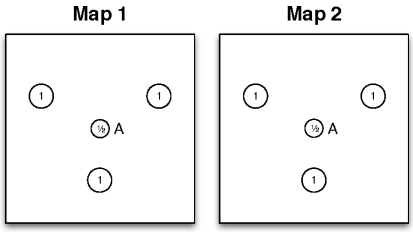
$$q_{j|i} = \frac{\sum_m \pi_i^{(m)} \pi_j^{(m)} \left(1 + \|y_i^{(m)} - \pi_j^{(m)}\|^2\right)^{-1}}{\sum_{m'} \sum_{i \neq k} \pi_i^{(m')} \pi_j^{(m')} \left(1 + \|y_i^{(m')} - \pi_j^{(m')}\|^2\right)^{-1}}$$

and we minimise the same objective function as in (5). [VdMH12] ◇

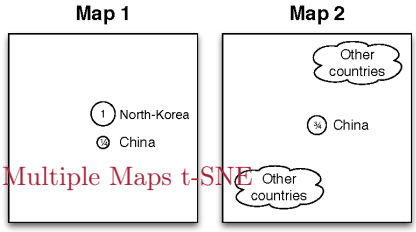
[Missing: Application of t-SNE in our Research, slides 54-62]



(a) Intransitive similarities.



(b) High centrality.



(c) Asymmetric similarities.

Fig. 25: [VdMH12]

2 Component Analysis

2.1 Canonical Correlation Analysis (CCA)

04.05.2020

In contrast to the previous methods, CCA extracts interesting components from the data, which is different from dimensionality reduction.

Setup. Assume that the measurements are linear superpositions of underlying factors (e.g. sources in EEG / MEG context):

$$x(t) = As(t) + \varepsilon,$$

where neither A nor the source s are known and ε is noise.

If we put some assumptions on A or s, we are able to factorise and invert the measured signal back to the source s, i.e. $s(t) = A^{-1}(x(t) - \varepsilon)$. There is a huge variety of methods employing different assumptions: PCA, CCA, ICA, JADE, TDSEP, SOBI, NGCA, CSP, SPOC, SSA, SCSA, MVARICA, CICAAR and more.

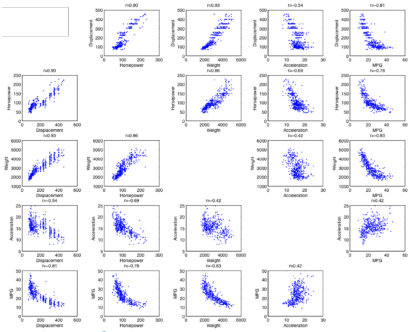
In CCA we assume there is a latent variable Z, which is hidden, and has an impact on variables $X \in \mathbb{R}^M$ and $Y \in \mathbb{R}^N$, which we can measure. How can we determine Z from X and Y, i.e. which representation of X and Y best represents Z? In CCA one aims to find the representation that maximises the correlation between X and Y.

Concretely, CCA finds projections $w_x \in \mathbb{R}^M$ and $w_y \in \mathbb{R}^N$ such that which solve [HOT36]

$$\arg\max_{w_x, w_y} w_x^\mathsf{T} X Y^\mathsf{T} w_y \quad \text{such that} \quad w_x^\mathsf{T} X X^\mathsf{T} w_x = w_y^\mathsf{T} Y Y^\mathsf{T} w_y = 1$$

Example 2.1.1 (CCA with cars)
Suppose the latent variable Z represents car types and the measurements X are displacement, horsepower and weights, whereas Y is acceleration and miles / gallon.

To analyse the data, we can look at the individual correlations between the variables (figure on the right). ◇



empirical cross-covariance matrices

Fig. 26: Some variables are highly correlated, but it is impractical to have to look at all 20 correlation plots to learn something about the data.

Assuming centered data, i.e. $\sum_i x_i = \sum_i y_i = 0$, we can compute the empirical cross-covariance matrices $C_{xy} = \frac{1}{N} X Y^\mathsf{T}$ and auto-covariance matrices $C_{xx} := \frac{1}{N} X X^\mathsf{T}$.

The Lagrangian is

$$L(w_x, w_y, a, b) := w_x^\mathsf{T} C_{xy} w_y - \frac{1}{2} a (w_x^\mathsf{T} C_{xx} w_x - 1) - \frac{1}{2} b (w_y^\mathsf{T} C_{yy} w_y - 1),$$

whose partial derivatives are

$$\frac{\partial L}{\partial w_x^\mathsf{T}} = C_{xy} w_y - a C_{xx} w_x, \quad \frac{\partial L}{\partial w_y^\mathsf{T}} = C_{xy}^\mathsf{T} w_x - b C_{yy} w_y = C_{yx} w_x - b C_{yy} w_y.$$

Setting the partial derivative and multiplying with w_x^T and w_y^T, respectively, yields

$$w_x^\mathsf{T} C_{xy} w_y = a w_x^\mathsf{T} C_{xx} w_x \quad \text{and} \quad w_y^\mathsf{T} C_{yx} w_x = b w_y^\mathsf{T} C_{yy} w_y$$

From the auto-covariance constraints $1 = w_x^\mathsf{T} C_{xx} w_x = w_y^\mathsf{T} C_{yy} w_y$ we obtain $a = b$, as $w_y^\mathsf{T} C_{yx} w_x = w_x^\mathsf{T} C_{xy} w_y$ by transposition.

Thus the Lagrangian becomes

$$L(w_x, w_y, a) := w_x^\mathsf{T} C_{xy} w_y - \frac{a}{2}\left(w_x^\mathsf{T} C_{xx} w_x - w_y^\mathsf{T} C_{yy} w_y\right),$$

and setting its partial derivatives to zero yields $C_{yx} w_x = a C_{yy} w_y$ and $C_{xy} w_y = a C_{xx} w_x$, which we can write as

$$\begin{bmatrix} 0 & C_{xy} \\ C_{yx} & 0 \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix} = a \begin{bmatrix} C_{xx} & 0 \\ 0 & C_{yy} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix}, \tag{6}$$

which is a generalised eigenvalue equation.

Among all eigenvectors (w_x, w_y), the solution of this problem is the one associated with the largest eigenvalue, which can be seen as follows (HA2-1a): Let $(w_x^{(i)}, w_y^{(i)})$ by the eigenvectors and $\lambda^{(i)}$ the corresponding eigenvalues. Multiplying the equation

$$\begin{bmatrix} 0 & C_{xy} \\ C_{yx} & 0 \end{bmatrix} \begin{bmatrix} w_x^{(i)} \\ w_y^{(i)} \end{bmatrix} = \lambda^{(i)} \begin{bmatrix} C_{xx} & 0 \\ 0 & C_{yy} \end{bmatrix} \begin{bmatrix} w_x^{(i)} \\ w_y^{(i)} \end{bmatrix}$$

by $[w_x^{(i)}, w_y^{(i)}]^\mathsf{T}$ on both sides yields

$$w_x^{(i)\mathsf{T}} C_{xy} w_y^{(i)} + w_y^\mathsf{T} C_{yx} w_x^{(i)} = \lambda (w_x^{(i)\mathsf{T}} C_{xx} w_x(i) + w_y^{(i)\mathsf{T}} C_{yy} w_y^{(i)})$$

The LHS is equal to $2 w_x^{(i)\mathsf{T}} C_{xy} w_y^{(i)}$ by transposition and the RHS is equal to $2\lambda^{(i)}$ due to the auto-covariance constraints, so the previous formula reads

$$w_x^{(i)\mathsf{T}} C_{xy} w_y^{(i)} = \lambda,$$

whose LHS is the quantity which CCA maximises. Thus

$$\max_{w_x^{(i)}, w_y^{(i)}} w_x^{(i)\mathsf{T}} C_{xy} w_y^{(i)} = \max_i \lambda^{(i)}.$$

But $\lambda^{\hat{i}}$, where $\hat{i}$ is the minimiser of the above problem, is the largest eigenvalue.

Furthermore (HA2-1b), (w_x, w_y) is a solution of CCA if and only if $(-w_x, -w_y)$ is a solution: $(-w_x)^\mathsf{T} C_{xy} (-w_y) = w_x^\mathsf{T} C_{xy} w_y$ and similarly for the auto-covariance constraints.

Example 2.1.2 (Cars - continued)
In the above example, we find $w_x = [0.0025, 0.0202, -0.000025]^\mathsf{T}$ and $w_y = [-0.17, -0.092]^\mathsf{T}$, yielding the projection on the right. After CCA, we have the components and can also look at other components, e.g. take the second component of each filter and project it to the second component (cf. figure on the right). ◇

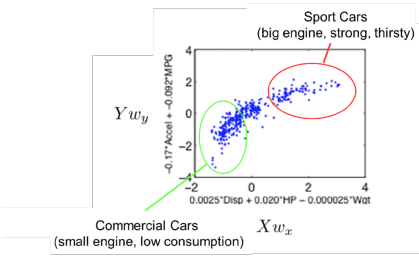


Fig. 27: todo

Remark 2.1.3 (History)
CCA is a fairly old basic well-known method, but there are several extensions to it. One old extension is CCA with more than two variables [Kettenring, 1971]. A more recent idea is kernel CCA (kCCA) [Akaho 2001], which also finds non-linear decencies and its applicable to high-dimensional data.

Recently, CCA become popular in machine learning as an objective function for kernel ICA [Bach 2002] and in Neuroscience. ◊

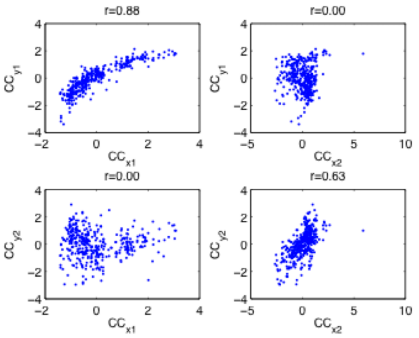


Fig. 28: todo

Example 2.1.4 (Improvement of standard CCA: kCCA) In certain scenarios, covariance matrices are too large to compute (e.g. bag-of-words feature space, which is potentially infinite dimensional). Also, CCA cannot capture non-linear dependencies. Kernel CCA operates on kernel of the data and not on covariance matrices and is thus faster and can also capture non-linear dependencies. ◇

Example 2.1.5 (Bag-of-words feature representation) A popular representation in text processing is Bag-of-words feature representation, where a word from a certain document is represented by counting the number of occurrences of the word. ◇

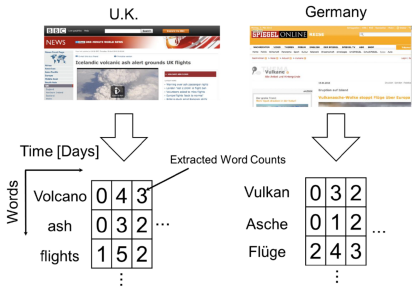


Fig. 29: Bag-of-words feature representation

If one uses kCCA, the kernel matrices, whose dimension is $N \times N$, where N is the number of data points. In standard CCA, the covariance matrices have dimension $d \times d$, where d is the dimension of the space.

For kCCA one can use the same intuition as for kPCA: any solution found by CCA has to lie in the subspace spanned by the data points.

Concretely, we show that there exists coefficient vectors $a_x, a_Y \in \mathbb{R}^N$ such that $w_x = Xa_x$ and $w_y = Ya_y$ (HA 2-2).

Proof. Towards contradiction, assume that $w_x = s_x + n_x$, where $s_x = Xa_x$ and n_x is noise term orthogonal to the data X and similarly for w_y. The CCA objective function then becomes

$$(s_x + n_x)^\mathsf{T} C_{xy}(s_y + n_y) = s_x^\mathsf{T} C_{xy} s_y + s_x^\mathsf{T} C_{xy} n_y + n_x^\mathsf{T} C_{xy} s_y + n_x^\mathsf{T} C_{xy} n_y.$$

As $n_x \perp X$, we have $n_x C_{xy} = \frac{1}{N} n_x X Y^\mathsf{T} = 0$ and, similarly, $C_{xy} n_y = XY^\mathsf{T} n_y = 0$, so the above term reduces to $s_x C_{xy} s_y$.

One similarly handles the auto-covariance constraints. □

This leads to the dual form of CCA: By using the dual variables a_x, a_y in (6), we obtain

$$\begin{bmatrix} 0 & C_{xy} \\ C_{yx} & 0 \end{bmatrix} \begin{bmatrix} Xa_x \\ Ya_y \end{bmatrix} = a \begin{bmatrix} C_{xx} & 0 \\ 0 & C_{yy} \end{bmatrix} \begin{bmatrix} Xa_x \\ Ya_y \end{bmatrix},$$

which is equivalent to

$$\begin{bmatrix} 0 & XY^\mathsf{T}Y \\ YX^\mathsf{T}X & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = a \begin{bmatrix} XX^\mathsf{T}X & 0 \\ 0 & YY^\mathsf{T}Y \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}.$$

Multiplying both sides by $\mathrm{diag}(X^\mathsf{T}, Y^\mathsf{T})$ yields

$$\begin{bmatrix} 0 & X^\mathsf{T}XY^\mathsf{T}Y \\ Y^\mathsf{T}YX^\mathsf{T}X & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = a \begin{bmatrix} X^\mathsf{T}XX^\mathsf{T}X & 0 \\ 0 & Y^\mathsf{T}YY^\mathsf{T}Y \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix},$$

which can be more compactly written as

$$\begin{bmatrix} 0 & AB \\ BA & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = a \begin{bmatrix} A^2 & 0 \\ 0 & B^2 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix},$$

with $A := X^\mathsf{T}X$ and $B := Y^\mathsf{T}Y$.

As above, one can show that the solution of the dual objective is given by the eigenvector associated with the largest eigenvalue: multiplying both sides with $[a_x, a_y]^\mathsf{T}$ yields

$$\begin{bmatrix} a_x^\mathsf{T} & a_y^\mathsf{T} \end{bmatrix} \begin{bmatrix} 0 & AB \\ BA & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = a \begin{bmatrix} a_x^\mathsf{T} & a_y^\mathsf{T} \end{bmatrix} \begin{bmatrix} A^2 & 0 \\ 0 & B^2 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix},$$

which is equivalent to

$$\begin{bmatrix} w_x^\mathsf{T} & w_y^\mathsf{T} \end{bmatrix} \begin{bmatrix} 0 & XY^\mathsf{T} \\ YX^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix} = a \begin{bmatrix} w_x^\mathsf{T} & w_y^\mathsf{T} \end{bmatrix} \begin{bmatrix} XX^\mathsf{T} & 0 \\ 0 & YY^\mathsf{T} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix}.$$

The solution of the primal problem can be recovered via $[w_x, w_y]^\mathsf{T} = [Xa_x, Ya_y]^\mathsf{T}$.

A sufficient representation of this subspace can be obtained by the inner products of all data points (linear kernels): $K_x \coloneqq X^\mathsf{T}X$, $K_y \coloneqq Y^\mathsf{T}Y$. The solution of CCA in the kernel space is obtained by solving the generalised eigenvalue problem

$$\begin{bmatrix} 0 & K_xK_y \\ K_yK_x & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \rho \begin{bmatrix} K_x^2 & 0 \\ 0 & K_y^2 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}.$$

The solutions in the input space can be recovered by $w_x = Xa_x$ and $w_y = Ya_y$.

The following code implements primal and dual CCA for low and high dimensional data, respectively.

```
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import utils
from scipy.linalg import eigh



def CCAprimal(X,Y):
    N = len(X)                          # len(X) = X.shape[0]

    Xc = X - X.mean(axis = 0)           # center data
    Yc = Y - Y.mean(axis = 0)

    Cxx = np.dot(Xc.T, Xc) / N
    Cyy = np.dot(Yc.T, Yc) / N
    Cxy = np.dot(Xc.T, Yc) / N
    Cyx = Cxy.T

    S = np.block([[Cxx*0, Cxy], [Cyx, Cyy*0]])

    D = np.block([ [Cxx, Cxy*0], [Cyx*0, Cyy]])

    vals, vecs = eigh(S, D)             # generalised
    eigendecomposition

    vec = vecs[:,np.argmax(vals)]   #sort by eigenvalues

    wx = vec[:X.shape[1]]
    wy = vec[X.shape[1]:]

```

```
    return wx, wy

X, Y   = utils.getdata()
p1, p2 = utils.plotdata(X,Y)

wx, wy = CCAprimal(X,Y)

p1,p2 = utils.plotdata(X,Y)
p1.arrow(0, 0, 1 * wx[0], 1 * wx[1], color = 'r', width =
    0.1)
p2.arrow(0, 0, 1 * wy[0], 1 * wy[1], color = 'r', width =
    0.1)
plt.show()

plt.figure(figsize=(6,2))
plt.plot(np.dot(X,wx))
plt.plot(np.dot(Y,wy))
plt.show()

### Dual formulation for higher dimensional data

def CCAdual(X,Y):
    N = len(X)

    X -= X.mean(axis = 0)
    Y -= Y.mean(axis = 0)

    Kx = np.dot(X, X.T)
    Ky = np.dot(Y, Y.T)

    Z = np.zeros((N, N))
    S = np.block([
        [Z, Kx.dot(Ky)],
        [Ky.dot(Kx), Z],
    ])
    D = np.block([
        [Kx.dot(Kx), Z],
        [Z, Ky.dot(Ky)],
    ])
    D = D + 0.001*D.std()*np.eye(len(D))    # to guarantee
    invertibility

    vals, vecs = eigh(S, D)

    vec = vecs[:,np.argmax(vals)]

    ax = vec[:N]
    ay = vec[N:]

    wx = X.T.dot(ax)
    wy = Y.T.dot(ay)

    return wx, wy

X,Y = utils.getHDdata()
wx,wy = CCAdual(X[:100],Y[:100])

utils.plotHDdata(wx,wy)
plt.show()

plt.figure(figsize=(6,2))
```

```
plt.plot(np.dot(X[100:],wx))
plt.plot(np.dot(Y[100:],wy))
plt.show()
```

Example 2.1.6 (Common semantic content extraction) One can image Z as being a document which is translated into many languages A, B and C. We aim to recover Z from A, B, C.

Visualising the principal directions, we can separate different categories. ◇

Remark 2.1.7 (CCA and least squares regression (HA 2-3)) Consider a supervised data set with centered inputs $X \in \mathbb{R}^{D\times N}$ and centered targets $y \in \mathbb{R}^N$. The least squares regression problem is

$$\max_{v\in\mathbb{R}^d} \|X^\mathsf{T} v - y\|^2.$$

If we now adopt a CCA viewpoint and view X and $y \in \mathbb{R}^{1\times N}$ as the two modalities of CCA, the first part of the solution w_x, will be a solution of least squares regression up to a scaling factor: the solution of least squares regression is $v = (XX^\mathsf{T})^{-1}Xy$, while the first line of the CCA generalised eigenvalue equation can be written as $Xyw_y = \lambda XX^\mathsf{T} w_x$, which is equivalent to $w_x = (XX^\mathsf{T})^{-1}Xy\left(\frac{w_y}{\lambda}\right)$ and $\frac{w_y}{\lambda}$ is a scalar multiple. ◊

Example 2.1.8 (Temporal kernel CCA (tkCCA))
If variables are coupled with delay (e.g. time series are correlated, but not simultaneously). To solve this, tkCCA shifts one variable relative to the other and then maximises correlation for (a sum over) all relative time lags. While (k)CCA finds canonical variates and correlation, tkCCA finds canonical convolution and correlogram (a visual way to show serial correlation in data that changes over time):

$$\underset{w_x(\tau),w_y}{\arg\max}\,\mathrm{Corr}\left(\sum_\tau w_x(\tau)^\mathsf{T} x(t-\tau), w_y^\mathsf{T} y(t)\right).$$

With $\tilde{X} := [X_{\tau_k}]_{k=1}^{T}{}^\mathsf{T}$ and $\tilde{w}_x := [w_x(\tau_k)]_{k=1}^{T}{}^\mathsf{T}$ (and w_y, Y similarly ??) we can rewrite the previous equation as

$$\underset{w_{\tilde{x}},w_y}{\arg\max}\,\mathrm{Corr}\left(w_{\tilde{x}}^\mathsf{T}\tilde{X}, w_y^\mathsf{T} Y\right). \quad ◇$$

Example 2.1.9 (Applying tkCCA to Neurovascular coupling)
There are two different modes to measure activity of neurons, one is EEG, which is a direct measurement of the electric activity and this usually has a very high temporal resolution, i.e. one can immediately measure the effects. In contrast, fMRI, which relies on the BOLD (blood oxygenation level dependent) signal, measure the relative change in blood oxygenation, i.e. the changes in metabolism in the brain (if cells are more active, they need more oxygen). fMRI has a high spatial but a low temporal resolution.

If one wants to combine EEG and fMRI in order to correlate results one can use tkCCA, as it allows to correlate data with delays. This

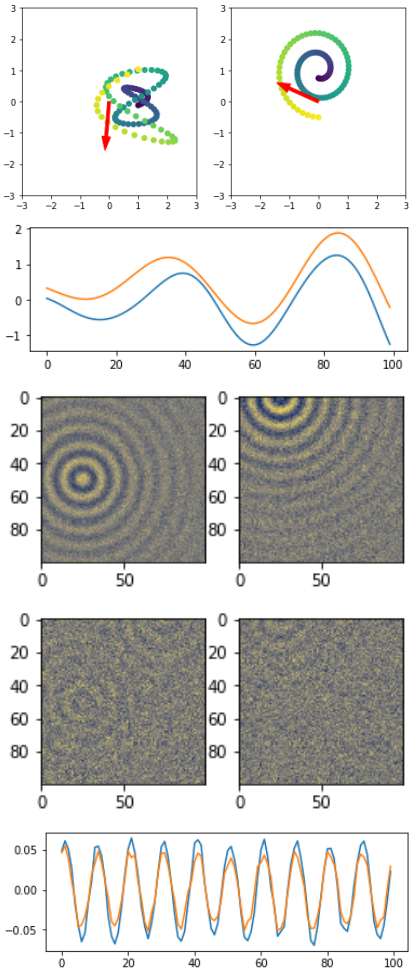


Fig. 30: TODO!

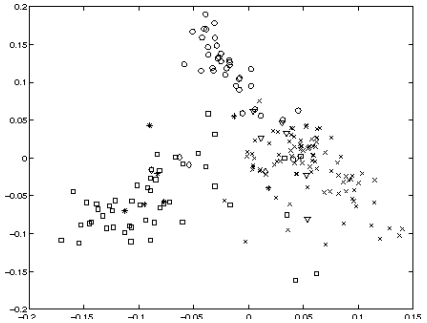


Fig. 31: First two canonical components of the bag of words kernels for the English (constitution) articles, as obtained by doing CCA with the other languages. Articles from different chapters are represented by a different symbol. [BC04]

Fig. 32: From left to right: neurophysical signal (?), spectogram of neural activity (frequency dependent HRF (hemodynamic response function, roughly measure blood oxygenation) (?), ?, ?.

yields to high-dimensional data X and Y with different dimensions and non-instantaneous couplings.

... some stuff (slides 25-29) ◇

Example 2.1.10 (Spatiotemporal dynamics of twitter replies) [BPH+12] Web content is often copied, repeated or rephrased ("trends"). This temporal structure contains important information. Canonical trend analysis exploits temporal structure to find trends and find web sources that precede / follow trends.

Can we, from the content of a web source, predict its spatiotemporal impact (how it will be retweeted, rephrased or copied and at what location) on a social network? Not only the time dimension (publishing time of a source) but also the location (where published and where retweeted) is important. For each news site $f \in \{1, \dots, F\}$ we extract bag-of-words features (for each news site, we have a BOW representation of the site at the time point t_0, $x_f(t = t_0))$ $X_f = [x_f(t = 1), \dots x_f(t = T)] \in \mathbb{R}^{W \times T}$ and the retweet locations $Y_f := [y_f(t = 1), \dots, y_f(t = T)] \in \mathbb{R}^{L \times T}$, whose values tell use e.g. how often a certain article has been retweeted.

[skipped slide 36, slide 37 not important] One downsamples geographic information, which works better in practice. [slide 38 not important]

For text data (X), Z can be interpreted as a topic. If we combine different words or dimensions in our BOW representation, this can be interpreted as a topic. E.g. if the mapping vector w_x combines the words "health care" and "Covid19", we know what the document / tweet is about: the Corona virus pandemic.

[Skipped slide 40 - 42]

We can now use a temporal embedding $\tilde{Y}_f := [(y_{f,\tau=k})_{k=1}^N]^\mathsf{T}$, where we put all the temporal data into one vector **(???)**, and apply standard CCA. But this will not be very effective because of the high dimensionality of $\tilde{Y}_f$.

Applying a linear kernel trick we get $w_y(\tau) = Y_{f,\tau}\alpha$, $w_x = X_f\beta$, which is very efficient for high-dimensional feature spaces. We maximise the objective function in its dual form

$$\begin{aligned}\mathrm{Corr}(\tilde{x}(t), \tilde{y}(t)) &= \frac{\sum_\tau (w_y(\tau)^\mathsf{T} Y_\tau)^\mathsf{T} X w_x}{\sqrt{\sum_\tau (w_y(\tau)^\mathsf{T} Y_\tau Y_\tau^\mathsf{T} w_y(\tau))\, w_x^\mathsf{T} X X^\mathsf{T} w_x}} \\ &= \frac{\alpha^\mathsf{T} K_{\tilde{Y}} K_X \beta}{\sqrt{\alpha^\mathsf{T} K_{\tilde{Y}}^2 \alpha \beta^\mathsf{T} K_X^2 \beta}},\end{aligned}$$

where $K_{\tilde{Y}} := \tilde{Y}^\mathsf{T}\tilde{Y}$ and $K_X := X^\mathsf{T}X$ are linear kernels.

The dual coefficients are solution to generalised eigenvalue equation

$$\begin{bmatrix} 0 & K_{\tilde{Y}} K_x \\ K_x K_{\tilde{Y}} & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} K_{\tilde{Y}} + I\kappa_y & 0 \\ 0 & K_X^2 + I\kappa_x \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Missing: Slide 45 **TODO**

We have mean word counts and want to know if they predict mean tweet frequency or the word count variance predicts the tweet variance.

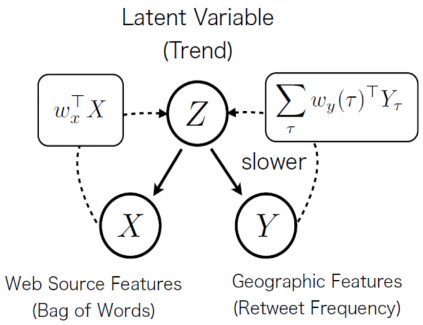


Fig. 33: Canonical trend model: We have data X, which is the web source features (bag of words of different web sources, e.g. newspapers, news sites or twitter accounts) and we have geographic features (retweet frequencies at different locations). We want to know the trend, which is a hidden variable, which affects both the content and the retweet frequency. We want to find a mapping from X to Z and from Y to Z.

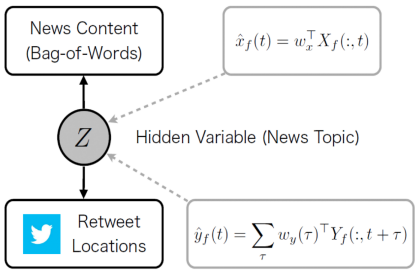


Fig. 34: **TODO???**

Hypothesis	**Objective??**
News content helps predict retweet frequency	$\arg\max_{w_y(\tau),w_x} \mathrm{Corr}(\hat{x}_f(t), \hat{y}_f(t))$
Mean word count predicts mean tweet frequency best	$w_x^\mathsf{T} = \frac{1}{N}\mathbb{1}_x$, $w_y^\mathsf{T} = \frac{1}{N}\mathbb{1}_y$
Word count variance predicts tweet variance	$\arg\max_{w_y(\tau)} w_y(\tau)^\mathsf{T}\tilde{Y}_f\tilde{Y}_f w_y(\tau)$, $\arg\max_{w_x} w_x^\mathsf{T} X X^\mathsf{T} w_x$ s.t. $w_y(\tau)^\mathsf{T} w_y(\tau) = w_x^\mathsf{T} w_x = 1$.

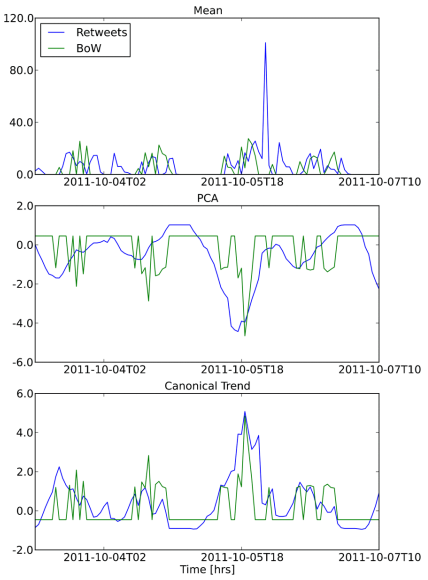


Fig. 35: Canonical trends are better fit for observations. [BPH+12]

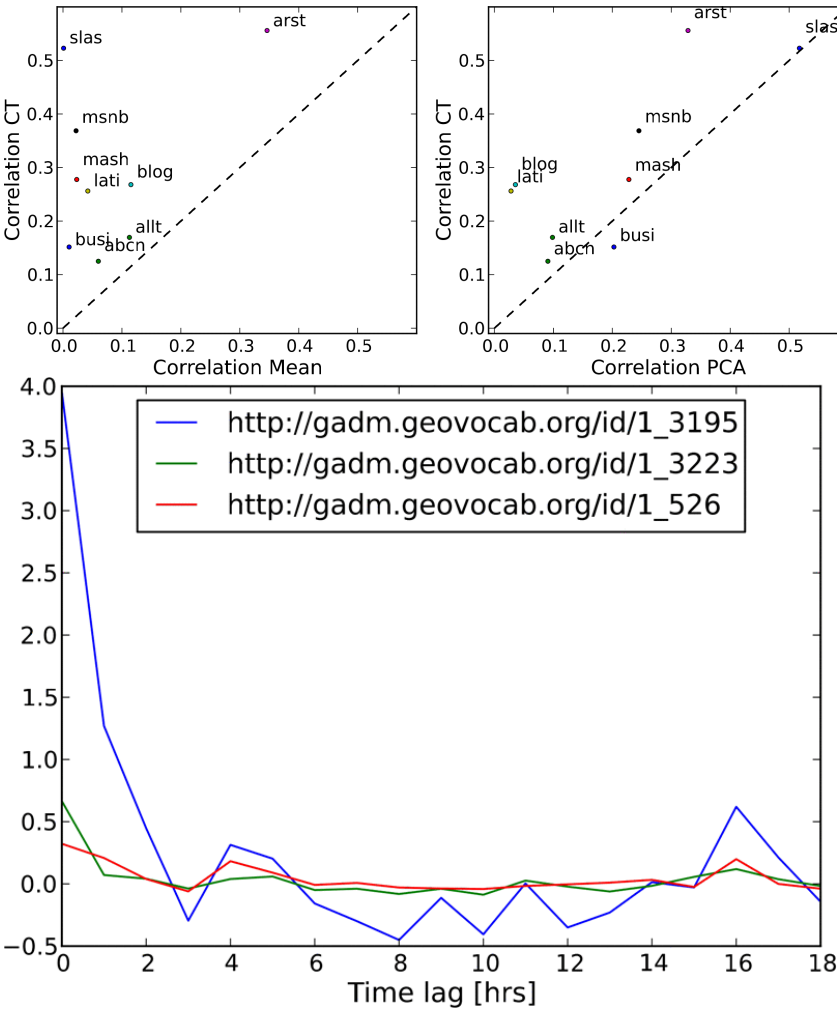


Fig. 36: **TODO First:** . **Second:** Correlation in absolute numbers (??) is much higher when applying CCA than when using PCA or the mean solution. **Third:** Excerpts from LA Times spatiotemporal response. The blue line of top is from California, which makes sense since this is an LA newspaper, while the green is New York and red is Ontario. The y-axis label should be $w_y(\tau)$. [BPH+12]

If we have a good model of the noise, we can further improve things.

[basically Skipped slide 51] ◇

Summary

- CCA finds projections for two data sets that maximise their correlation.
- kCCA extends CCA to potentially non-linear dependencies and makes CCA applicable to high dimensional data.
- tkCCA extends kCCA to data with non-instantaneous correlations and computes multivariate convolution from one modality to another.

2.2 Independent Component Analysis (ICA)

11.05.2020

Motivation: Blind source separation (BSS)

The problem ICA aims to solve can be modelled as a cocktail party problem: Assume you have multiple microphones (ears) and multiple speakers such that some microphones will record a superposition of some sound sources. One now wants to decompose / demix the superposed signal to e.g single out one speaker with many people talking simultaneously. This problem also appears in EEG and MEG and similar contexts.

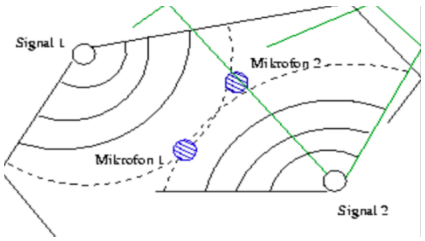


Fig. 37: The blind source separation problem. [Source?]

To fix ideas, consider microphones $x(t)$, which measures and unknown mixture of unknown sources, i.e. $x(t) = As(t)$, where x and s have the same dimension. Now assume statistical independence of the source signals (thus the name of ICA), which is a valid assumption if e.g. the sound sources differ (street noise, speech and music).

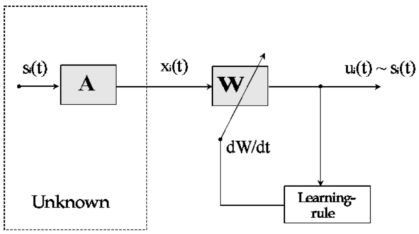


Fig. 38: **TODO????**

Our ansatz is to invert the mixing process A by learning W, given by $u(t) = Wx(t)$, and enforce the statistical independence of the unmixed signals $u(t)$.

The statistical independence of source translates to factorisation of the joint density: $p(u) = \prod_{i=1}^{n} p_i(u_i)$. Thus (due to independence) higher cross-moments vanish **(??)**.

We want to minimise the distance between distributions **(which ones?)**

$$D(W) := \int p(u) \log\left(\frac{p(u)}{\prod_{k=1}^{n} p_i(u_i)}\right) \mathrm{d}u$$

By Gram Chalier or Edgeworth expansion, one can expand $p_i(u_i)$ and plug them into the above formula. After tedious but straight forward calculation, we can find an explicit formula for $D(W)$.

TDSEP: BSS with temporal information

Another approach developed at TU Berlin assumes the same model $x(t) = As(t)$ and $u(t) = Wx(t)$ but also that s has significant autocorrelation (signal is correlated with signal at a previous point in time). In medical contexts, this is a valid assumption, as one e.g. expects high temporal correlation in EEG data.

Define the covariance matrices $V := \langle x_t x_t^\mathsf{T} \rangle$ and $V_\tau := \langle x_t x_{t-\tau}^\mathsf{T} \rangle$ over time $(\forall i \neq j)$**??)** and minimise

$$L(W) := \sum_{i \neq j} \langle u_i(t) u_j(t) \rangle^2 + \sum_{\{\tau\}} \langle u_i(t) u_j(t-\tau) \rangle^2,$$

(WHAT IS $\langle\cdot\rangle$???)where the first term measures the covariance between the estimated sources i and j. To enforce independence (stronger assumption that uncorrelatedness), the second term measures the covariance between time shifted sources. We can preform a simultaneous diagonalisation of $\{V, V_\tau, \ldots\}$ and thus not have to rely on gradient descent.

We need to whiten the data, which is a common feature of component analysis algorithms such as CCA, CSP, ICM, and brings the observed data into an uncorrelated form and projects in onto the unit ball. Here, the whitening transformation K is determined as the inverse square root of the covariance matrix:

$$K := \langle xx^\mathsf{T} \rangle^{-\frac{1}{2}} = (v\Lambda v^\mathsf{T})^{-\frac{1}{2}} \overset{??}{=} v\Lambda^{-\frac{1}{2}} v^\mathsf{T},$$

where $xx^\mathsf{T} = v\Lambda v^\mathsf{T}$ is the eigendecomposition of the covariance matrix. IN TDSEP, one simultaneously approximates the time-delayed covariance matrices, so one tries to find a matrix of eigenvectors Q, which simultaneously diagonalise the covariance matrices:

$$V_{\tau(z)} = \langle z_t z_{t-\tau}^\mathsf{T} \rangle = Q^\mathsf{T} V_{\tau(s)} Q = Q^\mathsf{T} \Lambda_\tau Q.$$

We can then determine the mixing matrix as $A = K^{-1}Q$. **(WHYYY?)**

In summary, we have seen that both methods enforce statistical independence, where the first approach uses high order statistics (expansions) and the second one second order statistics and temporal information.

Remark 2.2.1 (Nonlinear source separation)
ICA can be extended to a nonlinear scenario, where one assumes there exists a (nonlinear) function f such that $x(t) = f(A(s(t))$. ◊

Remark 2.2.2 (Reliability assessment)
Unsupervised learning techniques like ICA always return an answer/estimate found within their model class. One has to determine if the used model is appropriate and if/how one can assess the quality of the separation or specify error bars **(???)** to the estimates.

One way to asses reliability is to check reproducibility of the results. One can produce surrogate data sets which can be written as mixtures of independent source with the same mixing matrix A **(???)**: If the observed data $\{x(1), \ldots x(T)\}$ yields the mixing matrix $\hat{A}$, the produced data $\{x^{*i}(1), \ldots x^{*i}(T)\}$ for $i \in \{1, \ldots, k\}$, is explained by the matrices $\hat{A}^{*i}$.

1. Preform BSS with some ICA algorithm to obtain $Y = \hat{A}^{-1}X$.
2. From Y, produce surrogate data and then whiten theses data sets.
3. For each surrogate data set, preform BSS, yielding a set of k rotation matrices (as the data is already whitend **(Why does this follow???)**).
4. These matrices R can be decomposed into rotation angles via the matrix logarithm $a = \ln(R)$. **(log(R) is a matrix...)**
5. The standard deviations of the rotation angles define a separability

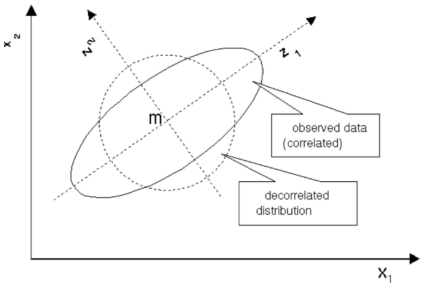


Fig. 39: TODO whitening transformation

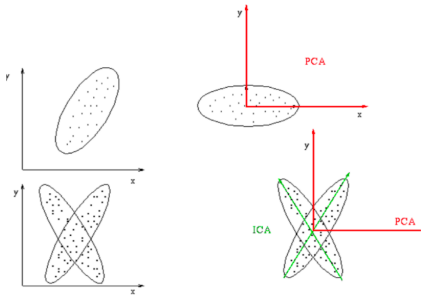


Fig. 40: PCA vs ICA.

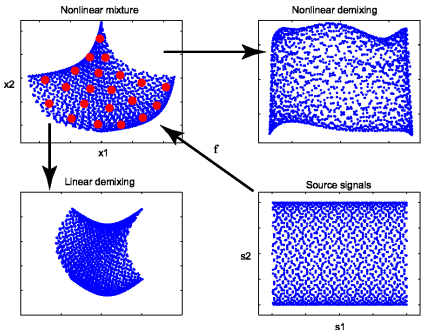


Fig. 41: BSS of nonlinearly distorted mixtures with kernel based learning methods [HZKM02].

matrix $S = \left(\sqrt{\langle a_{ij}^2 \rangle}\right)_{ij}$, which measures how unstable the estimated mixing matrix is with respect to a rotation in the plane spanned by the estimated components i and j.

6. The quantity $U_i := \max_j S_{ij}$ is the uncertainty of the estimated projection direction i, which approximates the RMSE (root-mean-square error).

Experimental results show that the (real) RMSE is nicely correlated to the (estimated) uncertainty. ◊

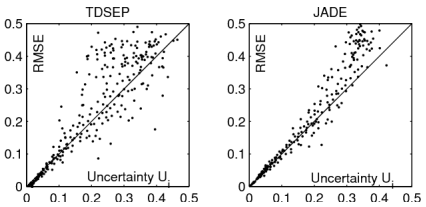


Fig. 42: The RMSE vs. the uncertainty estimate for the two used algorithms. For small values ($U \leqslant 0.1$) the uncertainty allows to predict the RMSE. [MZKM02]

Example 2.2.3 (Toy example: TDSEP vs JADE)
Consider a seven channel of two harmonic oscillations (like sin or cos), two speech signals, two white Gaussian noise processes and one uniformly distributed white noise. We investigate the source separation based on temporal decorrelation (TDSEP) and higher order statistics (JADE). The separability matrix for TDSEP indicates stable subspaces for (one-dimensional) speech signals and (two-dimensional) sinusodial signals, whereas for JADE also (one-dimensional) uniform white noise is stable.

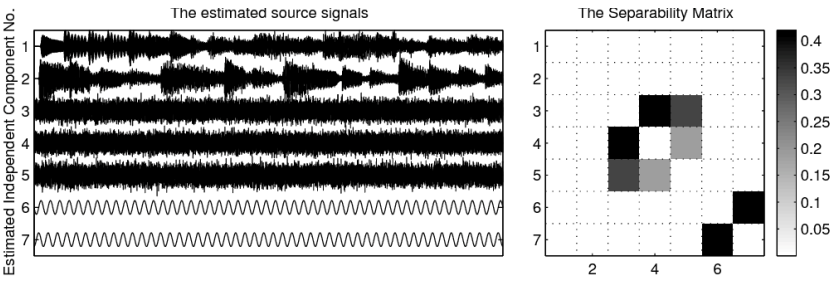


Fig. 5. Estimated sources and separability matrix using TDSEP (toy data set).

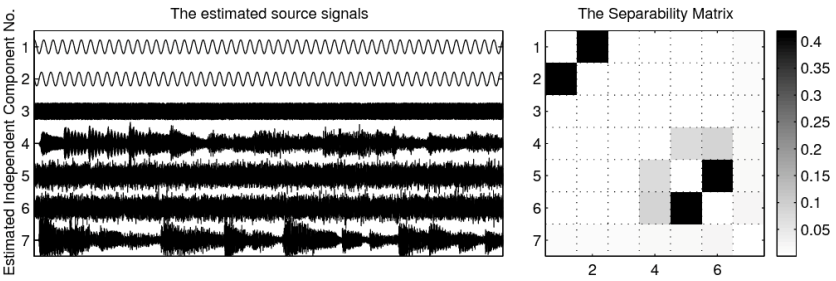


Fig. 6. Estimated sources and separability matrix using JADE (toy data set).

Fig. 43: [MZKM02]

Thus, TDSEP yields reliable estimates for audio sources (1, 2), while JADE yields reliable estimates of audio sources (4, 7) and the non-Gaussian random source (3). ◇

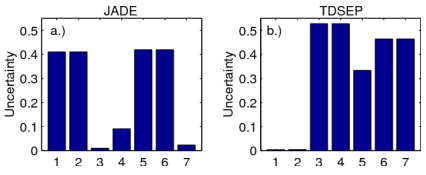


Fig. 44: TODO [MZKM02]

Thus to improve the separation performance one can

- Use different models on different sources, i.e. TDSEP on audio (maybe sin/cos) sources and JADE on non-Gaussian random source.
- use only "good" parts of a time series **(???)** [basically skipped]

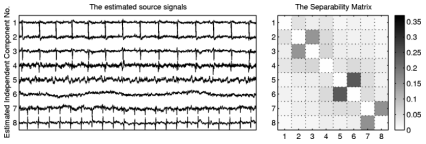


Fig. 45: [MZKM02] show that JADE separates cardiac signals of mother and fetus in an recording of a pregnant woman: the separability matrix has a block structure, which is of physiological relevance: it indicates independent multi-dimensional sub-

Example 2.2.4 (ICA of Non-invasively recorded DC-fields)
In [WZM+00] the goal is to discriminate "speakers in the brain" (analog

to cocktail party problem); identifying and extracting small brain signals despite the presence of noise. Relevant signals are often extremely weak compared to the noise (e.g. by a factor of 10000).

Applying ICA yields independent source and one hops that they are physiologically meaningful and one can not only extract the time series u, but also visualise the demixing matrix W or A.

Missing: slide 35, which is not important

Missing: images that are slides 36-41.

The image on slide 37 shows patients heads from above. Colour indicates, where important activity takes place.

Slide 38: analysing a single component.

Slide 39-40: TDSEP outperforms other methods. ◇

In summary,

- ICA demixes the data into "minimally statistically dependent" sources.
- Different measures of statistical dependence exists, yielding different algorithms such as JADE and TDSEP.
- Resampling method can be used to assess the quality of ICA projections.
- Application of ICA can reveal physiologically plausible stable subspaces.

Remark 2.2.5 (Invariances of ICA)

- **Scaling.** [Skipped:] Any scalar factor can be exchanged between each field pattern a_i and its source signal s_i without changing the resulting voltage of the electrodes:

$$x_i(t) = a_i s_i(t) = (\lambda a_i)\left(\frac{1}{\lambda} s_i(t)\right) = \tilde{a}_i \tilde{s}_i(t).$$

  Thus one is free to choose the sign and power of each signal

- **Permutation.** The numbering of the independent components is arbitrary (e.g. neural source in the brain do not have a canonical enumeration). ◊

Consequences of functional independence are statistical independence and uncorrelatedness even for time shifted signals **(???)** ($s_i(t)$ is uncorrelated with $s_j(t-\tau)$ for all τ).

Let x, y be independent random variables. Then

$$\mathbb{E}[g(x)h(y)] = \mathbb{E}[g(x)]\,\mathbb{E}[h(y)]$$

holds for all absolutely integrable functions g and h. If x and y are only uncorrelated, we have the above relation only for $g = h = \mathrm{id}$.

Definition 2.2.6 (Shannon / Differential entropy)
The Shannon-Entropy of a discrete random variable X is Shannon-Entropy

$$H(X) := -\sum_i x_i \log(x_i),$$

where a_i denotes the values of X and $x_i := P(X = a_i)$.

The differential entropy of a continuous random variable x with density p is

differential entropy

$$H(x) := -\int p(\xi)\log(p(\xi))\,\mathrm{d}\xi.$$

An ICA-algorithm seeks a linear invertible transformation **(B such that ??)** $y(t) = Bx(t)$ that minimises the mutual dependence between the components y_i. A suitable measure of dependence is the mutual information

mutual information

$$I(y_1, \ldots, y_N) := \sum_{i=1}^{N} H(y_i) - H(y),$$

which is minimal (zero) if and only if the variables are statistically independent and be seen as an analog to $D(W)$ from above. From $y = Bx$ we have $H(y) = H(x) + \log(|\det(B)|)$, so

$$I(y_1, \ldots, y_N) = \sum_{i=1}^{N} H(y_i) - H(x) - \log(|\det(B)|),$$

Only allowing transformations B yielding uncorrelated signals of variance, we obtain that $\det(B)$ does not depend on B but only on x, so

$$I(y_1, \ldots, y_N) = \sum_{i=1}^{N} H(y_i) + \text{const.},$$

so we have to minimise the entropy in each channel to minimise the mutual information. For a fixed mean and variance, the Gaussian distribution has the highest entropy, so ICA aims to find non-Gaussian projections.

Proof. (HA 3-1) We show that the random variable X with probability density function p solving

$$\max_X H(X) \quad \text{subject to} \quad \int_{\mathbb{R}} p(x)\,\mathrm{d}x = 1,\ \mathbb{E}[X] = 0,\ \mathrm{Var}[X] = \sigma^2$$

fulfills $X \sim \mathcal{N}(0, \sigma^2)$. In the following, we approximate the integrals by a Riemann sum of a regular partition G of $\mathbb{R}$. Each element of G is an interval of length δ represented by its center point x. This approximation lets us view the probability density function p as a collection of $|G|$ variables $(p(x))_{x\in G}$ and the objective can be rewritten as $H(X) = -\sum_{x\in G} p(x)\log(p(x))\delta$. Similarly, the equality constraints can be written as sums:

$$\sum_{x\in G} \exp(s(x))\delta = 1, \quad \sum_{x\in G} \exp(s(x))x\delta = 0, \quad \sum_{x\in G} \exp(s(x))x^2\delta = \sigma^2.$$

To handle the inequality constraint, we can apply the reparametrisation $p(x) = e^{s(x)}$ (The entropy is continuous with p optimising over the reduced class of functions, so we can ignore that $e^{s(x)} > 0$ and not $\geqslant 0$.) To handle the equality constraints, we will use the Lagrangian

$L(s,\lambda) \coloneqq L((s(x))_{x\in G}, \lambda_1, \lambda_2, \lambda_3)$, which is

$$-\sum_{x\in G}\exp(s(x))s(x)\delta + \lambda_1\left(\sum_{x\in G}\exp(s(x))\delta - 1\right) + \lambda_2\left(\sum_{x\in G}\exp(s(x))x\delta\right) + \lambda_3\left(\sum_{x\in G}\exp(s(x))x^2\delta - \sigma^2\right),$$

whose partial derivative is

$$\frac{\partial}{\partial s(x)}L(s,\lambda) = -\delta e^{s(x)}(s(x)+1) + \lambda_1\delta e^{s(x)} + \lambda_2\delta e^{s(x)}x + \lambda_3\delta e^{s(x)}x^2.$$

Setting this term to zero yields

$$-1 + \lambda_1 + \lambda_2 x + \lambda_3 x^2 = s(x),$$

implying

$$p(x) = \exp\left(-1 + \lambda_1 + \lambda_2 x + \lambda_3 x^2\right).$$

This form holds for any discretisation step, including those that converge to zero.

We want to show that $\lambda_3 = -\frac{1}{2\sigma^2}$, $\lambda_2 = 0$ and $\lambda_1 = 1 + \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$.

For $\lambda_3 < 0$ we have

$$\int_{\mathbb{R}} p(x)\,\mathrm{d}x = e^{\lambda_1 - 1}\int_{\mathbb{R}} e^{\lambda_2 + \lambda_3 x^2}\,\mathrm{d}x = e^{\lambda_1 - 1}\sqrt{\frac{\pi}{-\lambda_3}}\exp\left(-\frac{\lambda_2^2}{4\lambda_3}\right) \overset{!}{=} 1,$$

which is equivalent to

$$\exp\left(-\frac{\lambda_2^2}{4\lambda_3}\right) = e^{1-\lambda_1}\sqrt{\frac{-\lambda_3}{\pi}}. \tag{7}$$

and

$$\int_{\mathbb{R}} x\cdot p(x)\,\mathrm{d}x = e^{\lambda_1 - 1}\frac{\sqrt{\pi}}{2}\frac{\lambda_2}{(-\lambda_3)^{-3/2}}\exp\left(-\frac{\lambda_2^2}{4\lambda_3}\right) \overset{!}{=} 0. \tag{8}$$

Plugging (7) into (8) yields

$$0 = e^{\lambda_1 - 1}\frac{\sqrt{\pi}}{2}\frac{\lambda_2}{(-\lambda_3)^{-3/2}}e^{1-\lambda_1}\sqrt{\frac{-\lambda_3}{\pi}} = \frac{\lambda_2}{-2\lambda_3},$$

implying $\lambda_2 = 0$, which plugged into (7) yields

$$e^{\lambda_1 - 1} = \sqrt{\frac{-\lambda_3}{\pi}}. \tag{9}$$

Furthermore,

$$\begin{aligned}\int_{\mathbb{R}} x^2\cdot p(x)\,\mathrm{d}x &= e^{\lambda_1-1}\frac{\sqrt{\pi}}{4(-\lambda_3)^{5/2}}(\lambda_2^2 - 2\lambda_3)\exp\left(-\frac{\lambda_2^2}{4\lambda_3}\right)\\ &= e^{\lambda_1-1}\frac{\sqrt{\pi}}{2(-\lambda_3)^{3/2}} = \sqrt{\frac{-\lambda_3}{\pi}}\frac{\sqrt{\pi}}{2(-\lambda_3)^{3/2}} = -\frac{1}{2\lambda_3} \overset{!}{=} \sigma^2,\end{aligned}$$

yielding $\lambda_3 = -\frac{1}{2\sigma^2}$. Lastly, plugging this into (9) yields

$$\lambda_1 = 1 + \log\left(\sqrt{\frac{-\lambda_3}{\pi}}\right) = 1 + \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right).$$

(More derivations can be found here.) □

TODO: InfoMax and fastICA basically skipped.

Consider a joint probability distribution $p(x, y) := p(x)p(y|x)$, where

$$p(x) \sim \mathcal{N}(0, 1) \quad \text{and} \quad p(y|x) := \frac{\delta(y - x) + \delta(x + y)}{2}.$$

A useful property of linear component analysis for two-dimensional probability distributions is that the set of all possible directions to look for in $\mathbb{R}^2$ is $\{(\cos(\theta), \sin(\theta))^\mathsf{T} : \theta \in [0, 2\pi)\}$.

The projection of the random vector (x, y) on a particular component can therefore be expressed as a function of θ: $z(\theta) := x\cos(\theta) + y\sin(\theta)$. As a result, ICA in two dimensions is reduced to finding the values of the parameter $\theta \in [0, 2\pi)$ maximising an objective $J(z(\theta))$.

The principal components of $p(x, y)$ are the values of θ which maximise the variance of the projected data $z(\theta)$. We have

$$\begin{aligned}\mathbb{E}_{x,y}[z(\theta)(x, y)] &= \mathbb{E}_{x,y}[x] \cdot \cos(\theta) + \mathbb{E}_{x,y}[y] \cdot \sin(\theta) \\ &= \mathbb{E}_x[x] \cdot \cos(\theta) + \mathbb{E}_x\left[E_{y|x}[y]\right] \cdot \sin(\theta) = 0,\end{aligned}$$

where $\mathbb{E}_x[x] = 0$ as $p(x) \sim \mathcal{N}(0, 1)$ and $E_{y|x}[y] = 0$ as

$$\begin{aligned}E_{Y|x}[y] &= \int y \cdot p(Y = y|x)\,\mathrm{d}y \\ &= \int y \cdot \left[\frac{1}{2}\delta(y - x) + \frac{1}{2}\delta(y + x)\right]\mathrm{d}y = \frac{1}{2}x + \frac{1}{2}(-x) = 0,\end{aligned}$$

Thus

$$\begin{aligned}\mathrm{Var}[z(\theta)] &= \mathbb{E}_{x,y}[z(\theta)^2] \\ &= \cos^2(\theta)\,\mathbb{E}_{x,y}[x^2] + 2\cos(\theta)\sin(\theta)\,\mathbb{E}_{x,y}[xy] + \sin^2(\theta)\,\mathbb{E}_{x,y}[y^2] \\ &= \cos^2(\theta) + 2\cos(\theta)\sin(\theta)\,\mathbb{E}_x\left[x\,\mathbb{E}_{y|x}[y]\right] + \sin^2(\theta)\,\mathbb{E}_x[x^2] \\ &= \cos^2(\theta) + \sin^2(\theta) = 1,\end{aligned}$$

as $x \sim p(x)$ has variance 1 and

$$\begin{aligned}E_{Y|x}[y^2] &= \int y^2 \cdot p(Y = y|x)\,\mathrm{d}y \\ &= \int y^2 \cdot \left[\frac{1}{2}\delta(y - x) + \frac{1}{2}\delta(y + x)\right]\mathrm{d}y = \frac{1}{2}x^2 + \frac{1}{2}(-x)^2 = x^2.\end{aligned}$$

Thus the variance of all projections is equal and there are no principal components.

The independent components of $p(x, y)$ are the values of parameter θ that maximise the non-Gaussianity of $z(\theta)$. As a measure of non-Gaussianity we use excess kurtosis

$$\begin{aligned}\mathrm{kurt}[z(\theta)] &:= \frac{\mathbb{E}[(z(\theta) - \mathbb{E}[z(\theta)])^4}{\mathrm{Var}[z(\theta)]^2} - 3 = \mathbb{E}[z(\theta)^4] - 3 \\ &= \cos(\theta)^4\,\mathbb{E}_{x,y}[x^4] + 4\cos^3(\theta)\sin(\theta)\,\mathbb{E}_{x,y}[x^3y] \\ &\quad + 6\cos^2(\theta)\sin^2(\theta)\,\mathbb{E}_{x,y}[x^2y^2] \\ &\quad + 4\cos(\theta)\sin^3(\theta)\,\mathbb{E}_{x,y}[xy^3] + \sin(\theta)^4\,\mathbb{E}_{x,y}[y^4] - 3 \\ &= 3\cos^4(\theta) + 18\cos^2(\theta)\sin^2(\theta) + 3\sin^4(\theta) - 3 \\ &= 3\sin^2(2\theta).\end{aligned}$$

as $\mathbb{E}_x[x^4] = 3$ and, similarly to the above, $\mathbb{E}_{y|x}[y^3] = 0$ and $\mathbb{E}_{y|x}[y^4] = x^4$. The function attains its maximum, 1, for $\theta \in \left\{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right\}$.

Example 2.2.7 (Deriving a special case of FastICA (HW 3-3)) FastICA is a method for ICA that scales well in practice. Consider $x \in \mathbb{R}^d$ from some distribution p. We assume the data is centered and whitened: $\mathbb{E}[x] = 0$ and $\mathbb{E}[xx^\mathsf{T}] = I$. To extract and independent component, we would like to find a unit vector w such that the excess kurtosis of the projected data $w^\mathsf{T}x$ is maximised.

If w is a unit vector, the projection $w^\mathsf{T}x$ has zero mean and unit variance:

$$\mathbb{E}[w^\mathsf{T}x] = w^\mathsf{T}\,\mathbb{E}[x] = w^\mathsf{T}0 = 0$$

and

$$\mathrm{Var}[w^\mathsf{T}x] = \mathbb{E}[z^2] = \mathbb{E}[w^\mathsf{T}xx^\mathsf{T}w] = w^\mathsf{T}Iw = 1.$$

Thus

$$\mathrm{kurt}[z(\theta)] = \frac{\mathbb{E}[(w^\mathsf{T}x - \mathbb{E}[w^\mathsf{T}x])^4}{\mathrm{Var}[w^\mathsf{T}x]^2} - 3 = \mathbb{E}[(w^\mathsf{T}x)^4] - 3$$

The Lagrangian thus is

$$L(w, \lambda) \coloneqq \mathbb{E}[(w^\mathsf{T}x)^4] - 2\lambda(1 - \|w\|^2) = 4\,\mathbb{E}[x(w^\mathsf{T}x)^3] - 4\lambda$$

as expectation and differentiation can be exchanged. Setting this to zero yields

$$\lambda w = \mathbb{E}[x(w^\mathsf{T}x)^3].$$

The solution of the above equation cannot be found analytically. We will solve it via Newton's method, which assume that the equation is given as $F(w) = 0$ and uses the iteration $w_{k+1} \coloneqq w_k - J(w_k)^{-1}F(w_k)$, where J is the Jacobian of F.

Let $F(w) \coloneqq \mathbb{E}[x(w^\mathsf{T}x)^3] - \lambda w$. Then $J(w) = \mathbb{E}[3xx^\mathsf{T}(w^\mathsf{T}x)^2] - \lambda I$, so

$$w_{k+1} = w_k - (\mathbb{E}[3xx^\mathsf{T}(w_k^\mathsf{T}x)^2] - \lambda I)^{-1}(\mathbb{E}[x(w_k^\mathsf{T}x)^3] - \lambda w_k).$$

Under the decorrelation approximation

$$\mathbb{E}[xx^\mathsf{T}(w^\mathsf{T}x)^2] = \mathbb{E}[xx^\mathsf{T}] \cdot \mathbb{E}[(w^\mathsf{T}x)^2]$$

the Newton method reduces to

$$w_{k+1} = w_k - \frac{1}{3 - \lambda}(\mathbb{E}[x(w_k^\mathsf{T}x)^3] - \lambda w_k),$$

which reduces to

$$(3 - \lambda)w_{k+1} = (3 - \lambda)w_k - \mathbb{E}[x(w_k^\mathsf{T}x)^3] + \lambda w_k = 3w_k - \mathbb{E}[x(w_k^\mathsf{T}x)^3],$$

i.e.

$$w_{k+1} = \frac{\mathbb{E}[x(w_k^\mathsf{T}x)^3] - 3w_k}{\lambda - 3}$$

We can set $\gamma \coloneqq \frac{1}{\lambda - 3}$, which is some unimportant constant factor, as in every iteration we apply the normalisation $w_{k+1} \leftarrow \frac{w_{k+1}}{\|w_{k+1}\|}$. ◇

3 Kernel Machines

3.1 Structured Kernels / Inputs

18.05.2020

In this subsection we will look at structure kernels, which are an extension of standard (e.g. linear, polynomial, Gaussian) kernels to non-euclidean spaces. These kernels can be applied to DNA sequences, text or trees.

First, let us recap linear kernels.

From Linear to Kernel Models

Linear models are very common in machine learning. They can solve many different supervised tasks, such as (ridge) regression and classification (e.g. logistic regularisation, SVM) and unsupervised tasks such as component analysis (e.g. PCA, CCA, ICA). These tasks often come with desiderata (desirable properties) such as convexity (unique minima) and in come cases, closed form solutions.

Example 3.1.1 (Linear regression)
For training data $X := [x_1 \dots x_N]$ and targets $y = (y_k)_{k=1}^N$ the linear regression model $y = f(x) = b^\mathsf{T} x$ trained to minimise the least square error admits the closed form solution $b = (XX^\mathsf{T})^{-1}Xy$ (inverse covariance matrix × data × targets). ◇

To make the model nonlinear (and thus be able to solve nonlinear problems), we pass the data through a nonlinear feature map φ before applying the linear model: $f(x) = b^\mathsf{T}\varphi(x)$. To achieve this, we express the model in the span of the data:

$$b = \sum_{i=1}^{N} a_i \varphi(x_i).$$

We can then write the model and the training algorithm in a way that only involves the feature map through dot products:

$$f(x) = \sum_{k=1}^{N} a_k \varphi(x_k)^\mathsf{T}\varphi(x) \quad \text{and} \quad \frac{\partial E}{\partial a_i} = \sum_{k=1}^{N} 2(f(x_k) - y_k)\varphi(x_i)^\mathsf{T}\varphi(x_k),$$

where $E := \sum_{n=1}^{N}(f(x_n) - y_n)^2$.

We can now replace the dot products by a general nonlinear function $k\colon \mathbb{R}^d \to \mathbb{R}^d \to \mathbb{R}$, which represents similarity between the inputs:

$$k(x, y) := \varphi(x)^\mathsf{T}\varphi(y).$$

The model then becomes

$$f(x) = \sum_{i=1}^{N} a_i k(x_i, x).$$

In the case of a Gaussian kernel $k(x, z) := \exp\left(-\|c\|x - z\|^2\right)$ for $c > 0$, the model is a linear combination of "bumps", so by making c sufficiently small, any nonlinear function can be reconstructed, as long as the data points cover the space in a sufficient manner.

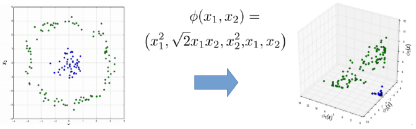


Fig. 46: When the feature map is higher dimensional, the expressive power of the model is increased: The two dimensional data on the right is not linearly separable (so we can not apply hard-margin SVM, soft-margin SVM will lead to result close to random guessing), but the mapped data is (only the first three components are plotted) as one allows for (nonlinear) multiplication of x_1 and x_2. (This map does not induce a polynomial kernel, but its variant $\varphi(x) := (x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$ does: $\varphi(x)^\mathsf{T}\varphi(y) = (\langle x, y \rangle + 1)^2$.)

We have seen that any feature map induces a kernel. By replacing the feature map with the kernel, we loose the view of the nonlinear model as a linear model in feature space: terms like $\langle w, \varphi(x) \rangle$ don't explicitly appear in the equations. For guaranteeing convergence and other regularisation properties of the learning algorithm **(???)**, we would like the reverse to be true.

Theorem 3.1.1: Mercer (1909)

If $k\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is symmetric and positive semidefinite (PSD), i.e. for all choices of $(x_i)_{i=1}^N \subset \mathbb{R}^d$ and $(c_i)_{i=1}^N \subset \mathbb{R}$ holds

$$\sum_{i,j=1}^{N} c_i c_j k(x_i, x_j) \geqslant 0,$$

then there exists a (potentially infinite dimensional) feature map φ such that $k(x, y) = \varphi(x)^\mathsf{T} \varphi(y)$.

Example 3.1.2 (Symmetric PSD kernels)
Kernels satisfying the above conditions include linear $(x^\mathsf{T} y)$ (which is equivalent to the linear model in the input space), polynomial $(x^\mathsf{T} y + a)^n$, Gaussian $\left(\exp\left(-\|c\|x - y\|^2\right)\right.$ and the t-Student kernel $(\|x - z\|^2 + a)^{-1}$, where $a \in \mathbb{R}$, $c > 0$ and $n \in \mathbb{N}$. ◇

In this lecture we will investigate how to build kernel for other types of data such as text, images, sequences, tree or graphs incorporating domain-specific knowledge. Text, for example, is translation invariant **(???)** and incorporating this information into the kernel can improve the statistical efficiency (from Wiki: a more efficient estimator or test needs fewer observations than a less efficient one to achieve a given performance) of the model, which is useful if we have few data points.

Example 3.1.3 (Convolution kernel)
For signals $x = (x_k)_{k \in \mathbb{Z}}$ and $y = (y_k)_{k \in \mathbb{Z}}$ their (discrete) convolution is

$$(x * y)_t := \sum_{\tau \in \mathbb{Z}} x(\tau) y(t - \tau).$$

The convolution kernel is convolution kernel

$$k(x, y) := \|x * y\|_2^2 = \sum_{t \in \mathbb{Z}} ((x * y)_t)^2,$$

which is positive semidefinite: for signals $x_1, \ldots x_n$ and coefficients $c_1, \ldots, x_n \in \mathbb{R}$ we have, by the symmetry of convolution,

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) = \sum_{i,j=1}^{n} c_i c_j \sum_{\tau, \tau' \in \mathbb{Z}} x_i(\tau) x_j(t - \tau) x_j(\tau') x_i(t - \tau').$$

By the substitution $s := t - \tau - \tau'$ we obtain

$$\sum_{i,j=1}^{n} c_i c_j \sum_{\tau, \tau' \in \mathbb{Z}} x_i(\tau) x_j(s + \tau') x_j(\tau') x_i(s + \tau), \tag{10}$$

which is equal to

$$\sum_{s\in\mathbb{Z}} \left(\sum_{i=1}^{n} \sum_{\tau\in\mathbb{Z}} c_i x_i(\tau) x_i(s+\tau) \right)^2 \geqslant 0.$$

From (10) we can infer that

$$\varphi(x) := \left(\sum_{\tau\in\mathbb{Z}} x(\tau) x(s+\tau) \right)_{s\in\mathbb{Z}} \overset{?}{=} x * x$$

is a feature map for k. ◇

Representing strings

Definition 3.1.4 (Alphabet, string)
An alphabet $\mathcal{A}$ is a finite set of discrete symbols. A string is concatenation of symbols from an alphabet $\mathcal{A}$. By $\mathcal{A}^L$ we denote all strings of length L and define $\mathcal{A}^* := \bigcup_{L\in\mathbb{N}} \mathcal{A}^L$.

Example 3.1.5 (Alphabet)
An alphabet can be a set of states $\{S_1, \ldots, S_d\}$ of a machine, a nucleotide basis $\{G, A, T, C\}$ or language text $\{a, b, c, \ldots, A, B, C, \ldots, 0, 1, 2, \ldots\}$. ◇

We now show four approaches to designing a string kernel. The example strings will be $x := abcceabd$ and $z := abacccbe$.

(1) Counting the number of matching symbols, yields the kernel

$$k\colon \mathcal{A}^L \times \mathcal{A}^L \to \mathbb{R},\ (x, z) \mapsto \sum_{i=1}^{L} \mathbb{1}(x_i = z_i).$$

For the example strings we get $k(x, z) = 4$ by the following table.

x	a	b	c	c	e	a	b	d
y	a	b	a	c	c	c	b	e
$\mathbb{1}(x_i = z_i)$	1	1	0	1	0	0	1	0

This kernel induces the feature map

$$\varphi(x) := (\mathbb{1}(x_i = a))_{\substack{a\in\mathcal{A},\\ i\in\{1,\ldots L\}}} \in \{0,1\}^{L\times|\mathcal{A}|}.$$

Indeed, for $x, y \in \mathcal{A}^L$, we have

$$\begin{aligned} \langle \varphi(x), \varphi(y) \rangle &= \operatorname{Tr}(\varphi(x)^\mathsf{T} \varphi(y)) \\ &= \sum_{\ell=1}^{L} \sum_{a\in\mathcal{A}} \mathbb{1}(x_\ell = a)\, \mathbb{1}(y_\ell = a) = \sum_{\ell=1}^{L} \mathbb{1}(x_\ell = y_\ell). \end{aligned}$$

(2) Counting the number of matching symbols with *some* shift tolerance, yields the alignment kernel

$$k(x, z) := \sum_{i=1}^{L} \alpha\, \mathbb{1}(x_i = z_{i-1}) + \beta\, \mathbb{1}(x_i = z_i) + \gamma\, \mathbb{1}(x_i = z_{i+1})$$

for $\alpha, \beta, \gamma \in \mathbb{R}$. For the example strings we get $k(x, z) = 10$ for $\alpha, \gamma = 1$ and $\beta = 2$ by the following table.

x	a	b	c	c	e	a	b	d
y	a	b	a	c	c	c	b	e
$\mathbb{1}(x_i = y_{i-1})$	-	0	0	0	0	0	0	0
$\mathbb{1}(x_i = y_i)$	1	1	0	1	0	0	1	0
$\mathbb{1}(x_i = y_{i+1})$	0	0	1	1	0	0	0	-

This kernel is PSD for $\alpha, \gamma = 1$ and $\beta = 2$ but not for $\alpha, \beta, \gamma = 1$:
For $x^{(1)}, \dots, x^{(N)} \in \mathcal{A}^L$ and $c_1, \dots, c_N$ the term $\sum_{i,j=1}^N c_i c_j k(x^{(i)}, x^{(j)})$ is equal to

$$\sum_{i,j=1}^{N} c_i c_j \left(\sum_{\ell=1}^{L} \mathbb{1}(x_\ell^{(i)} = x_{\ell-1}^{(j)}) + 2 \cdot \mathbb{1}(x_\ell^{(i)} = x_\ell^{(j)}) + \mathbb{1}(x_\ell^{(i)} = x_{\ell+1}^{(j)}) \right)$$

$$= \sum_{\ell=1}^{L} \sum_{k \in \mathcal{A}} \sum_{i=1}^{N} c_i \, \mathbb{1}(x_\ell^{(i)} = k) \sum_{j=1}^{N} c_j \left[\mathbb{1}(x_{\ell-1}^{(j)} = k) + 2 \cdot \mathbb{1}(x_\ell^{(j)} = k) + \mathbb{1}(x_{\ell+1}^{(j)} = k) \right]$$

$$= \textbf{TODO}.$$

3 Counting the number of matching symbols with *full* shift tolerance, yields the "Bag-of-Words"-kernel

$$k(x, z) := \sum_{i,j=1}^{L} \mathbb{1}(\{x_i = z_j\}) = \sum_{w \in \mathcal{A}} \#_w(x) \cdot \#_w(z), \qquad (11)$$

where $\#_w(x)$ is the number of times the symbol w appears in the string x. For the example strings we get $k(x, z) = 15$ by the following table.

x	a a	b b	c c	d	e
y	a a	b b	c c c		e
	4	4	6	0	1

Consequently, the feature map for this kernel is

$$\varphi \colon \mathcal{A}^L \to \mathbb{R}^{|\mathcal{A}|}, \; x \mapsto \left(\sum_{i=1}^{L} \mathbb{1}(x_i = w) \right)_{w \in \mathcal{A}},$$

as

$$\varphi(x)^\mathsf{T} \varphi(z) = \sum_{w \in \mathcal{A}} \left(\sum_{i=1}^{L} \mathbb{1}(x_i = w) \right) \left(\sum_{i=1}^{L} \mathbb{1}(z_i = w) \right)$$
$$= \sum_{w \in \mathcal{A}} \#_w(x) \cdot \#_w(z).$$

4 Counting the number of matching subsequences of symbols of length n ("n-grams") yields the n-gram kernel

$$k(x, z) := \sum_{i=1}^{L+1-n} \mathbb{1}(\{(x_i, \dots, x_{i+n-1}) = (z_i + \dots, z_{i+n-1})\})$$

For $n = 2$ we get $k(x, z) = 1$ by the following table.

x	a b	b c	c c	c e	e a	a b	b d
y	a b	b a	a c	c c	c c	c b	b e
	1	0	0	0	0	0	0

One can also combine those kernels.

The n-gram approach is always applicable and best suitable for analysis of strings with unknown structure, e.g. DNA sequences, network attacks or binary data.

The tokenisation approach, where the string is broken up into tokens, is suitable for analysis strings with known structure, e.g. natural language text (tokens are the words), tokenised data, log files.

tokenisation approach

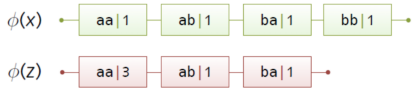


Fig. 47: The preprocessing step for the "Bag-of-Words" kernel.

Remark 3.1.6 (Implementing string kernels efficiently)
Evaluating the "Bag-of-Words" kernel (11) is computationally expensive. This can be reduced by building a preliminary data structure for each string, i.e. a *sorted* list where every string and its count is stored. One then just has to go through both lists and multiply the entries until one is empty. ◊

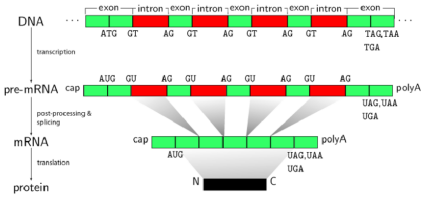


Fig. 48: Biosynthesis. [SWSR08]

Example 3.1.7 (String kernels for genomic data [SSP+07])
In protein biosynthesis, the DNA is reduced to mRNA via extraction of the introns, from which then the proteins are encoded. To understand what parts of the DNA encoded which proteins, one needs to detect the splice sites (transitions from exon to intron).

The idea behind Splice Site Prediction is build a kSVM that predicts slice sites. The kSVM is applied as a sliding window through large nucleotide sequences. A weighted degree kernel, which is a weighted sum of n-gram kernels is used to detect if a splice site is currently at the center of the input window:

$$k\colon \mathcal{A}^L \times \mathcal{A}^L \to \mathbb{R},\ (x,z) \mapsto \sum_{n=1}^{d} b_n \sum_{i=1}^{L+1-n} \mathbb{1}\left((x_k)_{k=i}^{i+n-1} = (z_k)_{k=i}^{i+n-1}\right),$$

where $\mathcal{A} := \{G, A, T, C\}$ and $b_1, \ldots, b_n \geqslant 0$.

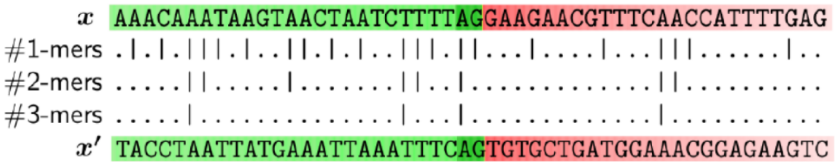


Fig. 49: The different rows indicate if there are shared n-grams ("#n-mers") between the two sequences x and x'.

A computational trick to apply the weighted degree kernel is to identify maximum blocks of common subsequences.

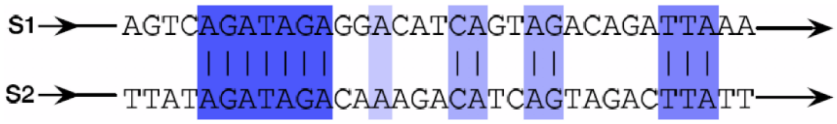


Fig. 50: Illustration of just looking for maximum blocks of common subsequences.

The kernel is positive semidefinite: for ... and coefficients $c_1, \ldots, c_n \in \mathbb{R}$

we have

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) = \sum_{i,j=1}^{n} c_i c_j \sum_{m=1}^{M} b_m \sum_{\ell=1}^{L+1-m} \mathbb{1}\left(u_{\ell,m}(x_i) = u_{\ell,m}(x_j)\right)),$$

which is equal to

$$\sum_{m=1}^{M} b_m \sum_{k \in A^m} \sum_{\ell=1}^{L+1-m} \left(\sum_{i=1}^{n} \mathbb{1}(u_{\ell,m}(x_i) = k) \right)^2 \geqslant 0.$$

Thus for $M = 1$, the feature map is

$$\varphi(x) := \sqrt{b_1}(\mathbb{1}(u_i(x) = k))_{i \in \{1,\dots,L\}, k \in A} \in \mathbb{R}^{L \times 4}.$$

Similarly, for $M = 2$ and $b_1 = 0$, $b_2 = 1$ (only examining sequences of length two) the feature map is

$$\varphi(x) := (\mathbb{1}(u_i(x) = k))_{i \in \{1,\dots,L\}, k \in A^2} \in \mathbb{R}^{L \times 16}. \qquad \diamond$$

Kernels for Trees

Definition 3.1.8 (Parse tree)
A tree $x = (V, E, v^*)$ is an acyclic graph (V, E) rooted at $v^* \in V$.

A parse tree $x \in T$ is a tree deriving from a grammar, such that each node $v \in V$ is associated with a production rule $p(v)$.

This is a common structure in several application domains, e.g. natural language processing, compiler design and many more.

The idea is to decompose this parse tree as we would like to build a kernel that can compare two different parse trees. We characterise the parse trees in terms of their subtrees.

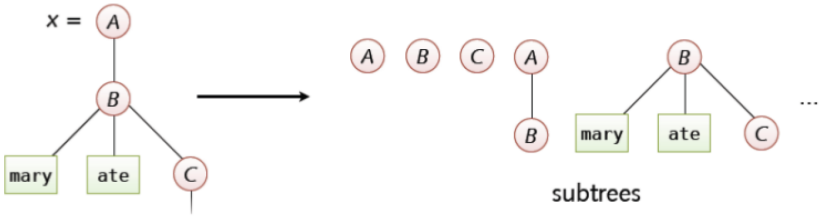


Fig. 52: The subtrees of different sizes of the tree considered above.

A feature map mapping trees to $\mathbb{R}^{|T|}$ is

$$\varphi \colon T \to \mathbb{R}^{|T|}, \; x \mapsto (\#_t(x))_{t \in T},$$

where $\#_t(x)$ counts the occurrences of a subtree t in the tree x. This feature map induce a parse tree kernel

$$k \colon T \times R \to \mathbb{R}, \; (x, y) \mapsto \langle \varphi(x), \varphi(y) \rangle = \sum_{t \in T} \#_t(x) \cdot \#_t(y),$$

which can be considered an analogon to the Bag-of-Words kernel (11). To make computing k less expensive (there are exponentially many subtrees),

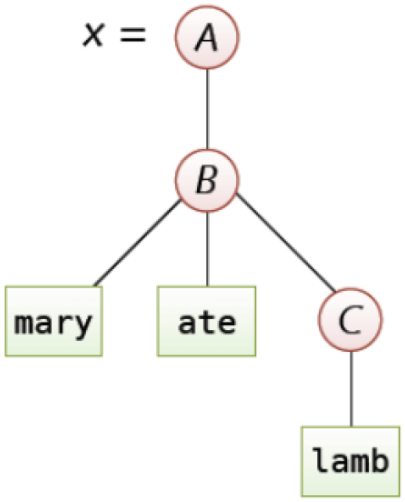


Fig. 51: Tree representation of sentences derived from a grammar: Parse tree for "mary at lamb" with production rules $p_1 \colon A \to B$, $p_2 \colon B \to$ "mary" "ate" C and $p_3 \colon C \to$ "lamb": The tree has three nodes, A, B and C. A produces the sentence, B produces the subject, verb and object of the sentence and C produces the object.

we can, for each pair (v, w) determine shared subtrees at v and w such that

$$k(x, y) = \sum_{\substack{v \in V_x \\ w \in V_y}} c(v, w),$$

where

$$c(v,w) = \begin{cases} 0, & \text{if } p(v) \neq p(w) \text{ (different production)}, \\ 1, & \text{if } |v| = |w| = 0 \text{ (leaf node)}, \\ \prod_{i=1}^{|v|} \big(1 + c(v_i, w_i)\big), & \text{otherwise.} \end{cases}$$

where i-th child of a node $v \in V$ is v_i and $|v|$ denotes the number of children of v.

An efficient implementation using dynamic programming may look like this: starting with the smallest subtrees first, one progressively "grows" them. At some point, one can identify common structure and update the count to incorporate the count for the number of subtrees common the two matched trees. Proceeding iteratively, until reaching the full tree will lead to the final kernel score $k(x, y)$ reached in $O(|V_x|, |V_y|)$.

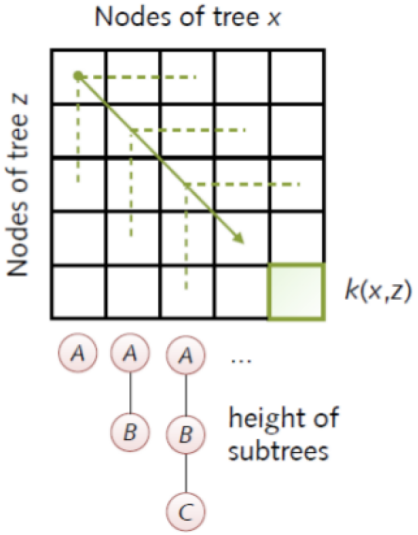


Fig. 53: An efficient implementation using dynamic programming visualised.

Model-induced kernels

model-induced kernels

Another type of structured kernels are model-induced kernels, whose aim it is to not to build a new kernel directly on the input data, but to extract a kernel from the local response of the existing model $f_\theta(x)$ to its learning parameter vector $\theta \in \mathbb{R}^{|\theta|}$, e.g. define

$$\psi \colon X \to \mathbb{R}^{|\theta|}, \; x \mapsto \nabla_\theta f_\theta(x)$$

and construct the kernel as

$$k(x, y) = \kappa(\psi(x), \psi(y)),$$

where κ is a standard kernel (e.g. linear, etc.).

The fisher kernel uses a generative model as an existing model.

Example 3.1.9 (Fisher kernel)
The Fisher kernel is a structured kernel induced by a probability model $p_\theta(x)$. Let $x \in X$ and suppose $p_\theta : X \to \mathbb{R}$ is a generative model (e.g. hidden Markov model) for the data parametrised by $\theta \in \mathbb{R}^h$. Defining

$$G_x := \frac{\partial}{\partial \theta} \log(p_\theta(x)),$$

the Fisher kernel is

$$k(x, y) := G_x^\mathsf{T} \left(\mathbb{E}_{z \sim p_\theta}[G_z G_z^\mathsf{T}]\right)^{-1} G_y.$$

While it is mainly used to extract a feature map of fixed dimensions from structured data on which a structured probability model readily exists (e.g. a hidden Markov model), the Fisher kernel can in principle also be derived for simpler distributions such as the multivariate Gaussian distribution. We here consider the covariance matrix Σ to be fixed such

that the only effective parameter on which the Fisher kernel is based, is the mean μ.

We thus have

$$\begin{aligned} G_x &= \frac{\partial}{\partial \mu}\left(-\frac{1}{2}\log\left((2\pi)^d \det(\Sigma)\right) - \frac{1}{2}\left((x-\mu)^\mathsf{T}\Sigma^{-1}(x-\mu)\right)\right) \\ &= -\frac{1}{2}\left(-\Sigma^{-1}x - \Sigma^{-1}x + 2\Sigma^{-1}\mu\right) = \Sigma^{-1}(x-\mu). \end{aligned}$$

and thus, with $A = \Sigma^{-1}$

$$\begin{aligned} k(x,y) &= (x-\mu)^\mathsf{T} A \left(\mathbb{E}_{Z\sim\mathcal{N}(\mu,\Sigma)}\left[A(Z-\mu)(Z-\mu)^\mathsf{T}A\right]\right)^{-1} A(y-\mu) \\ &= (x-\mu)^\mathsf{T}\left(\mathbb{E}_{Z\sim\mathcal{N}(\mu,\Sigma)}\left[(Z-\mu)(Z-\mu)^\mathsf{T}\right]\right)^{-1}(y-\mu) \\ &= (x-\mu)^\mathsf{T}\left(\mathrm{Var}_{Z\sim\mathcal{N}(\mu,\Sigma)}\left[Z\right]\right)^{-1}(y-\mu) = (x-\mu)^\mathsf{T}\Sigma^{-1}(y-\mu). \end{aligned}$$

As Σ^{-1} is positive definite, there exits a unique square root $\Sigma^{-\frac{1}{2}}$. The feature map for the above kernel thus is $\varphi(x) \coloneqq \Sigma^{-\frac{1}{2}}(x-\mu)$. ◇

Diffusion kernels

A diffusion kernel is not specifically designed to accommodated special types of inputs like strings or tree, but to introduce prior knowledge into the kernel, specifically the local geometry of the data given by some graph.

Diffusion kernels assume that the input domain is discrete and that the local geometry is given by a graph $G(V,E)$, where each node represents a data point. The diffusion kernel defines the generator matrix H by

$$H_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ -\deg(v_i) & \text{if } v_i = v_j, \\ 0 & \text{otherwise,} \end{cases}$$

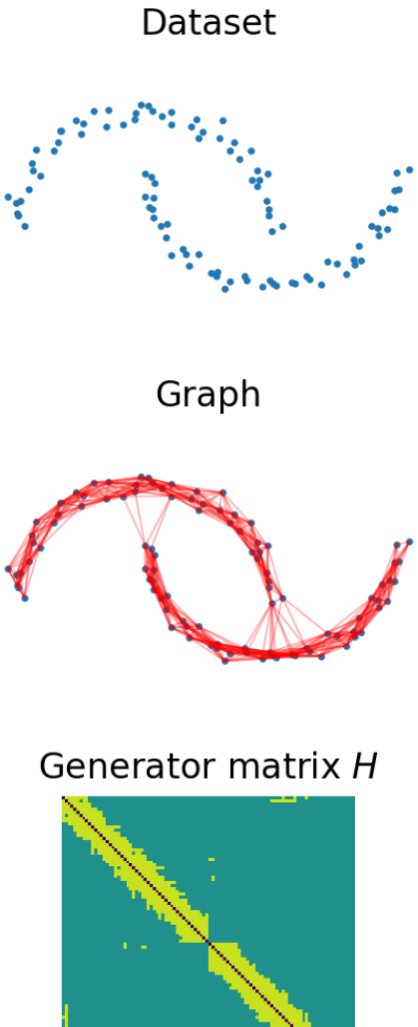
where $\deg(v)$ is the number of edges from v, and then diffuses the graph signal by matrix exponentiation:

$$k(x_i, x_j) \coloneqq [e^{bH}]_{i,j},$$

where b is a kernel hyperparameter, which determines the strength of the diffusion (larger b = more diffusion).

The diffusion kernel is particularly useful for semi-supervised learning, where on only has few labelled data and lots of unlabelled data, which form an underlying graph structure. If the data has a manifold structure, we can diffuse the label into the manifold using the generator matrix. It is also useful or for unsupervised analyses such as (spectral) clustering.

Example 3.1.10 (Diffusion kernel on two dimensional data)
Consider the data set with manifold structure and then build a graph connecting neighbouring data points. One can see that there are spurious connections between the two separate structures.



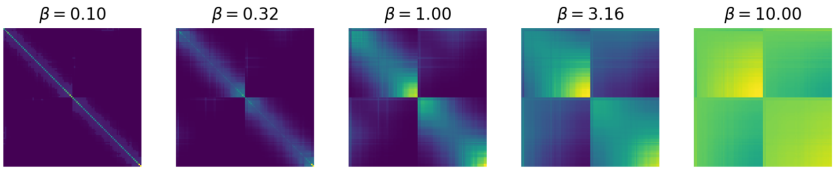


Fig. 55: The results show that the higher β, the higher the similarity between points of the same cluster and that if β is too high, leaks start to appear because points on different manifolds become similar, which is the effect because of the spurious connections.

One can see that the plots are roughly divided into four parts, the two structures are mostly similar to themselves and data points of different manifolds are not similar; the matrix is strongly diagonal. When increasing the diffusion parameter β, points that are a bit further apart on the same manifold start becoming similar. ◇

In summary, structured kernels can be used to predict data, which is not in $\mathbb{R}^d$, e.g. sequences of symbols or trees and can be designed to incorporate prior knowledge, e.g. positional invariance or diffusion on a graph. Once the kernel structure has been defined, most popular learning algorithms such as least square regression, SVMs, PCA, CCA and any other linear model, can be used.

3.2 Structured Prediction / Outputs

25.05.2020

Standard machine learning models $f : X \to Y$ produce simple outputs such as real numbers / vectors (in (multivariate) regression $Y = \mathbb{R}^c$). In this subsection we present a framework where Y can have a complex structure, e.g. a sequence or tree.

Example 3.2.1 (Handwriting: Sequential)
The unstructured output version is to split an image containing a handwritten word into pieces containing the individual characters and then classify them. It has several limitations: one has to exactly know the segmentation of the word and by interpreting the characters individually, their concatenation might not yield a meaningful word.

The structured output version addresses these issues by taking as input not a single character but an image containing the whole sequence of characters and then outputting a word directly. The advantage of this approach is that no segmentation is needed and the output can be constraint to only include meaningful words. ◇

Example 3.2.2 (Context free grammar parsing: Recursive)
One wants to predict the grammar of an input sentence. A grammar is usually represented as a tree which defines the different parts of the sentence and how they relate to each other. Trees are not a simple structure, which can't be predicted directly with standard machine learning techniques, thus we need structured output.

One wants to find the best parsing tree of a sentence. The output domain

contains possible parsing trees for that sentence, good ones and bad ones, and the learning algorithm / prediction is able to determine which one is best. ◇

Example 3.2.3 (Bilingual word alignment: Combinatorial)
The input consists of the two sentences in two different languages. The goal of structured predictions is to find a matching between the two sentences (typically nonlinear with the flow of the sentence because of the different grammatical rules in different languages). The set of all bipartite graphs connecting at least two lists of items is a combinatorial structure, which can't be predicted by standard machine learning techniques. ◇

Example 3.2.4 (Label sequence learning)
Given an observation sequence of nucleotides that defines a gene, one likes to predict the classes (exon and intron) which decide the content of resulting protein. Instead of predicting single elements, one would like to predict the whole sequence. This allows us to add as a constraint that two adjacent elements have a high probability of being of the same class, avoiding e.g. exons appearing in the middle of introns. ◇

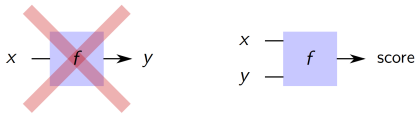


Fig. 56: Standard machine learning vs structured outputs.

In the last section we have seen that the standard machine learning model is capable of handling structured inputs, but we cannot do the same for the outputs. Instead we consider both the input x and the prediction y to be inputs of a function $f\colon X \times Y \to \mathbb{R}$, which outputs a score for how good the prediction y is for the input x. We can not just evaluate the function and get the prediction as an output but instead we have to solve an optimisation problem over all possible outputs $y \in Y$:

$$y|x := \arg\max_{y \in Y} f(y|x). \tag{12}$$

This approach is more flexible as x and y can be structured, but is also is computationally more expensive.

Kernel-based structured prediction

There are different approaches to structured prediction based on (12), one of which is energy-based learning, but in this lecture we will look at kernel-based structured prediction, which defines a kernel function in the joint input / output domain:

$$k\colon (X \times Y) \times (X \times Y) \to \mathbb{R}\,.$$

When the kernel is positive semi-definite, it will induce a feature map (with typically finite-dimensional codomain)

$$\varphi\colon X \times Y \to \mathbb{R}^h$$

for $h \in \mathbb{N}$, on which we can build the scoring model

$$f(y \mid x) := w^{\mathsf{T}} \varphi(x, y)$$

for some $w \in \mathbb{R}^h$, which we have to learn. (If φ maps into a matrix space, replace the last term by a scalar product.)

Structured output can be seen as a framework which extends standard prediction methods but also contains simpler models as special cases.

Example 3.2.5 (Multiclass classification)
Let $X := \mathbb{R}^d$, $Y := \{1, \dots, C\}$ and k some input kernel on X which feature map $\varphi\colon X \times Y \to \mathbb{R}^h$. The structured kernel

$$K\colon (X \times Y) \times (X \times Y),\ (x_1, y_1), (x_2, y_2) \mapsto k(x_1, x_2) \cdot \mathbb{1}(\{y_1 = y_2\})$$

is positive definite: for all $x_1, \dots, x_n \in X$, $y_1, \dots, y_n \in Y$ and $c_1, \dots, c_n \in \mathbb{R}$ the term $\sum_{i,j=1}^n c_i c_j K(x_i, x_j)$ is equal to

$$\begin{aligned} &\sum_{c=1}^{C} \left(\sum_{i=1}^{n} c_i \, \mathbb{1}(y_i = c) \varphi(x_i) \right)^{\mathsf{T}} \left(\sum_{j=1}^{n} c_j \, \mathbb{1}(y_j = c) \varphi(x_j) \right) \\ = &\sum_{c=1}^{C} \left\| \sum_{i=1}^{n} c_i \, \mathbb{1}(y_i = c) \varphi(x_i) \right\|_2^2 \geqslant 0. \end{aligned}$$

Consequently, K induces the feature map

$$\psi(x, y) := (\varphi(x) \cdot \mathbb{1}(\{y = 1\}), \dots, \varphi(x) \cdot \mathbb{1}(\{y = C\}))^{\mathsf{T}} \in \mathbb{R}^{h \times C}$$

and thus the decision function becomes

$$y|x = \arg\max_{y=1}^{C} w^{\mathsf{T}} \psi(x, y) = \arg\max_{c} w_c^{\mathsf{T}} \varphi(x),$$

where $w_c \in \mathbb{R}^C$ is a weight vector and c is the index of the class. One obtains, as an equivalent formulation, a set of weights, one per class, that one multiplies with the feature representation of the input and then take the weights corresponding to the class such that the dot product is maximal. ◇

Learning a large-margin model

How can we train such models? Consider a structured output model

$$f(y|x) := w^{\mathsf{T}} \varphi(x, y),$$

where the prediction is given by $y|x = \arg\max_{y \in Y} f(y|x)$. Let $((x_k, y_k))_{k=1}^N \subset X \times Y$ be the training set.

The learning problem can be formulated as a large margin problem: We want to find the largest margin (w, ξ) by solving (define $\Psi_{n,y} := \varphi(x_n, y_n) - \varphi(x_n, y)$ for convinience)

$$\min_{w,\xi} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \xi_n \quad \text{s.t.} \quad w^{\mathsf{T}} \Psi_{n,y} \geqslant 1 - \xi_n,\ \xi_n \geqslant 0\ \forall_{n=1}^N \forall_{y \neq y_n}, \quad (13)$$

where the ξ_k are slack terms (SVM) which account for misclassification or noise in the data and $C > 0$ is a regularisation parameter (large C $\implies$ hard margin).

We now look at how costly it is to solve this optimisation problem. The optimisation problem is too large, as there are exponentially many constraints. We thus only incorporate the few violated constraints. A possible procedure is

1. Begin with a small set of wrong labelings.

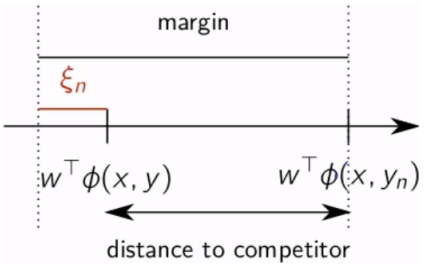


Fig. 57: Geometrical interpretation of the large-margin model. Both x's should be replaced by x_n's.

2. Solve the thus reduced optimisation problem, ignoring all other constraints by solving $\arg\max_{y \in Y} w^\mathsf{T} \varphi(x, y)$.
3. Find the new labelings that violate constraints
4. Add constraints and return to step two.

This will converge **(WHYY??)** and typically only a subset of the exponentially many constraints will be included. Especially the very trivial labelings, which are clearly wrong, will not be added before we reach convergence, so we never have to express them explicitly in the optimisation problem.

We will later see that this procedure only works for the primal formulation and not for the dual.

How can we find violated constraints? We see that if $w^\mathsf{T}\varphi(x_n, y)$ is very large, it will definitely violate the constraint so we look for

$$\arg\max_{y \in Y} w^\mathsf{T} \varphi(x_n, y).$$

This can be difficult due to the large number of possible y. But if the structure if φ permits, we can use dynamic programming to quickly find such labelling:

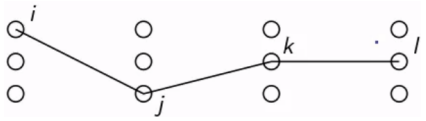


Fig. 58: Think of this diagram as a Markov chain. Each circle is a state and a column of circles is the set of possible states. Different columns represent the state at different time steps. The path shown with lines depicts a possible sequence of states. Our goal is to find the best possible sequence of states and the variable a represents the cost or reward of transitioning between the different states.

Example 3.2.6 (Dynamic programming (Viterbi))
Assume the output y is a sequence and that the function $w^\mathsf{T}\varphi(x, y)$ decomposes into a sum of subfunctions involving only adjacent terms of the sequence. In the case of the figure of the right, the function to optimise takes the form

$$w^\mathsf{T}\varphi(x, (y_i, y_j, y_k, y_\ell) = a_i + a_{ij} + a_j + a_{jk} + a_k + a_{k\ell} + a_\ell,$$

but there e.g. can not be any terms depending on i and k.

The maximisation of that function can be achieved by "pushing" the max function "into" the sum:

$$\begin{aligned} &\max_{i,j,k,\ell} w^\mathsf{T}\varphi(x, (y_i, y_j, y_k, y_\ell) \\ =\ &\max_i \left(a_i + \max_j \left(a_{i,j} + a_j + \max_k \left(a_{jk} + a_k + \max_\ell a_{k,\ell} + a_\ell \right) \right) \right), \end{aligned}$$

which can be solved in linear (each max takes constant time) instead of exponential time. The maximising elements can then be recovered by backtracking the max operation (i.e. this works also for arg max, not only for max).

In order to design feature maps that obey this subfunction conditions there are two approaches.

1. **(Sum structure)** If the map φ fulfills the condition

$$\varphi(x, (y_i, y_j, y_k, y_\ell) = \varphi_{ij}(x) + \varphi_{jk}(x) + \varphi_{k\ell}(x),$$

so does $w^\mathsf{T}\varphi$:

$$\begin{aligned} w^\mathsf{T}\varphi(x, (y_i, y_j, y_k, y_\ell) &= w^\mathsf{T}\varphi_{ij}(x) + w^\mathsf{T}\varphi_{jk}(x) + w^\mathsf{T}\varphi_{k\ell}(x) \\ &=: a_{ij} + a_{jk} + a_{k\ell}. \end{aligned}$$

(2) **(Concatenation structure)** If we instead concatenate the components (which yields a larger feature map), i.e.

$$\varphi(x,(y_i,y_j,y_k,y_\ell) = [\varphi_{ij}(x),\varphi_{jk}(x),\varphi_{k\ell}(x)]\,,$$

the function $w^\mathsf{T}\varphi$ fulfills the condition (where w is as large as the feature map)

$$\begin{aligned} w^\mathsf{T}\varphi(x,(y_i,y_j,y_k,y_\ell) &= w_{[1,\dots,h]}^\mathsf{T}\varphi_{ij}(x) + w_{[h+1,\dots,2h]}^\mathsf{T}\varphi_{jk}(x) \\ &\quad + w_{[2h+1\dots,3h]}^\mathsf{T}\varphi_{k\ell}(x) \\ &=: a_{ij} + a_{jk} + a_{k\ell}. \end{aligned}$$

Both forms lead to a sum-decomposition. The feature map or kernel can be designed to introduce desired prior knowledge and invariances. ◇

Example 3.2.7
Consider output sequences to predict to be of the type $y \in \{-1,1\}^L$ and the feature map

$$\varphi(x,y) \coloneqq [x \odot y, 3\cdot(y_1,\dots,y_{L-1}) \odot (y_2,\dots,y_L)].$$

The structured output model aims to solve

$$\max_{y\in\{\pm1\}^L} w^\mathsf{T}\varphi(x,y) \tag{14}$$

for $w \in \mathbb{R}^{2L-1}$.

Assume $L=3$ and that the current parameter is $w \coloneqq (1,\dots,1)$ and we receive the input $x=(1,-1,1)$. We then have

$$\begin{aligned} (14) &= \max_{y\in\{\pm1\}^L} y_1 - y_2 + y_3 + 2y_1y_2 + 2y_2y_3 \\ &= \max_{y_1\in\{\pm1\}} \left\{ y_1 + \max_{y_2\in\{\pm1\}} \left\{ 2y_1y_2 - y_2 + \max_{y_3\in\{\pm1\}} \{2y_2y_3 + y_3\} \right\} \right\}. \end{aligned}$$

This can be solved iteratively, using Viterbi's procedure:

$$\max_{y_3\in\{\pm1\}} \{2y_2y_3 + y_3\} = 2y_2 + 1$$

(so $y_3 = 1$) and

$$\max_{y_2\in\{\pm1\}} 2y_1y_2 - y_2 + 2y_2 + 1 = \max_{y_2\in\{\pm1\}} 2y_1y_2 + y_2 + 1 = 2y_1 + 2$$

(so $y_2 = 1$) and, finally,

$$\max_{y_1\in\{\pm1\}} y_1 + 2y_1 + 2 = 5,$$

so $y = (1,1,1)$. ◇

Dual formulation

For the cases where we can not use the procedure incorporating dynamic programming outlined above, we now introduce an extension of the primal formulation, which will allow us to have the feature map be expressed as dot products. This is useful when we don't want to compute the feature

map explicitly but access the data and compare it via a kernel function directly.

The dual formulation of (13) is

$$\max_{a} \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} - \frac{1}{2} \sum_{n_1,n_2} \sum_{\substack{y_1 \neq n_1 \\ y_2 \neq n_2}} a_{n_1,y_1} a_{n_2,y_2} \cdot \langle \Psi_{n_1,y_1}, \Psi_{n_2,y_2} \rangle$$

$$\text{such that } \forall_{n=1}^{N} \forall_{y \neq y_n} : a_{n,y} \geqslant 0 \quad \text{and} \quad \sum_{y \neq y_n} a_{n,y} \leqslant C.$$

Proof. The Lagrangian is

$$L(w, \xi, a, b) := \frac{\|w\|_2^2}{2} + C \sum_{n=1}^{N} (1 - b_n) \xi_n + \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \left(1 - \xi_n - w^{\mathsf{T}} \Psi_{n,y}\right).$$

We have

$$\frac{\partial}{\partial w} L(w, \xi, a, b) = 0 \iff w = \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Psi_{n,y}$$

and

$$\frac{\partial}{\partial \xi_n} L(w, \xi, a, b) = 0 \iff C - \sum_{y \neq y_n} a_{n,y} - \beta_n = 0$$

for $n \in \{1, \ldots, N\}$. The dual formulation thus is

$$\max_{a,b \geqslant 0} \frac{1}{2} \left\| \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Psi_{n,y} \right\|_2^2 + C \cancel{\sum_{n=1}^{N} \xi_n}$$
$$+ \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \left(\cancel{1 - \xi_n} - \left(\sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Phi_{n,y} \right)^{\mathsf{T}} \Psi_{n,y} \right) - \cancel{\sum_{n=1}^{N} b_n \xi_n},$$

which is

$$\max_{a \geqslant 0} \frac{1}{2} \left\| \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Psi_{n,y} \right\|_2^2$$
$$+ \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \left(1 - \left(\sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Psi_{n,y} \right)^{\mathsf{T}} \Psi_{n,y} \right),$$

which is equivalent to

$$\max_{a \geqslant 0} -\frac{1}{2} \left\| \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y} \Psi_{n,y} \right\|_2^2 + \sum_{n=1}^{N} \sum_{y \neq y_n} a_{n,y}.$$

Since $\sum_{y \neq y_n} a_{n,y} = C - \beta_n$ and β_n, we can rewrite it as $\sum_{y \neq y_n} a_{n,y} \leqslant C$.□

Unlike in the primal formulation, there is no simple procedure to reduce the large number of constraints.

Loss functions

Another extension of the structured SVM is the incorporation of a loss function. So far, we have implicitly use a 0-1-loss function with slack variables: if $y|x_n \neq y_n$, then the prediction is wrong, but we don't know "how wrong" it is. Instead, we can introduce a loss function $\ell(y_1, y_2)$

on the labellings, which can determine e.g. how many segments in the sequence are wrong or missing, or how different the segments are. With this more sophisticated loss function, we can penalise very wrong predictions more than slightly wrong ones.

There are two variants of the SVM with this loss function

1. **Margin rescaling.** We replace the size of the margin, which is one in the primal formulation, by the loss function:

$$w^\mathsf{T}\Psi_{n,y} \geqslant \ell(y, y_n) - \xi_n$$

Thus for very incorrect predictions, the margin and the slack variable will have to be larger than for slightly incorrect ones.

2. **Slack rescaling.** Here, the loss functions is incorporated as the denominator of the slack function:

$$w^\mathsf{T}\Psi_{n,y} \geqslant 1 - \frac{\xi_n}{\ell(y, y_n)},$$

thus forcing ξ_n to be large if $\ell(y, y_n)$ is large, which in turn enlarges $C\sum_{n=1}^{N}\xi_n$. So if $\ell(y, y_n)$ is large, we treat the output y_n causing the wrong prediction in a hard-margin setting, whereas if the prediction is mildly wrong, the soft-margin is active.

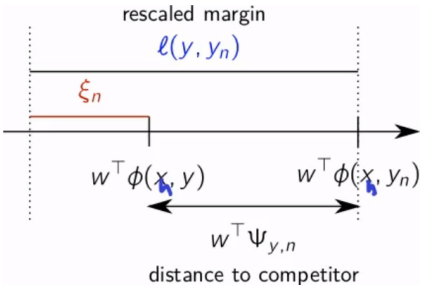


Fig. 59: Geometric interpretation of margin rescaling.

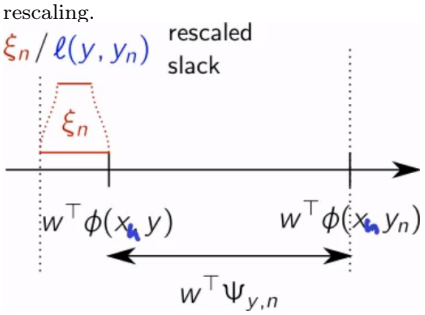


Fig. 60: Geometric interpretation of slack rescaling.

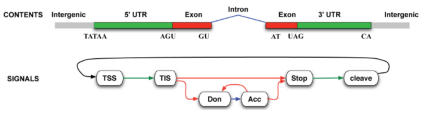


Fig. 61: The lower diagram shows the transition policy between the different states, which our structured outputs model will learn to implement.

Example 3.2.8 (Application: Gene sequence tagging)
We have a gene sequence and aim to assign a tag to each element of the sequence. Each element of the sequence is a state and between different elements of the sequences there a transitions. Typically they are Markovian segment level, so it switches from one segment to the other in a Markovian way and non-Markovian within segments in order to stay in one segment longer. The function f represents the score assigned to a certain predicted sequence y:

$$f(y) := \underbrace{\sum_{j=1}^{J-1} S_{GT}(f_j^{GT}) + \sum_{j=2}^{J} S_{AG}(f_j^{AG})}_{\text{Splice signals}}$$
$$x + \underbrace{\sum_{j=1}^{J-1} S_{L_I}(p_{j+1} - q_j) + \sum_{j=1}^{J} S_{L_E}(q_j - p_j)}_{\text{Segment lengths}},$$

which is composed of sums corresponding to different elements of the sequence of pairs of elements of the sequence, which implements the constraint on the relation between the different elements of the sequence or the preference of a given element of the sequence.

We have to tune free parameters (in the functions S_{GT}, S_{AG}, S_{LE} and S_{L_I}) by solving a linear program using a training set with known splice forms.

[Missing: Picture (slide21) is it relevant?] ◇

Hidden Markov Model

In this lecture we have looked at structured prediction, but there is another model, which also has structure and has been around for several decades:

the Hidden Markov Model (HMM). It is a joint probability model over inputs and outputs, whee the probability distribution factorises into adjacent terms:

$$p(x, y) = \prod_i p(x_i|y_i)p(y_i|y_{i-1}),$$

which can be seen as satisfying the Markov property, where p is the joint distribution over inputs x and output sequence y. The log probability $\log(p(x, y))$ decomposes into a sum of contributions that involve only adjacent terms. Thus we can use dynamic programming techniques we have used in structured prediction to infer the most likely outputs.

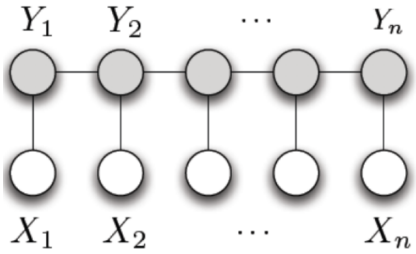


Fig. 62: Source?

Remark 3.2.9 (Structured prediction vs. HMM)
In HMM, the model is learned in a fully unsupervised manner. Once the model has been learned, the procedure for sequence prediction is fully determined. In structured prediction, however, the choice of feature map and loss function give more flexibility into the structure of the model and the model parameter $w \in \mathbb{R}^h$ can be actively optimised for best performance on the supervised task. ◊

In summary,

- Structured output learning enables the prediction of structured objects such as sequences or trees.
- Structured prediction operates very different from standard machine learning models by assigning matching scores to input/output pairs instead of predicting the output directly.
- The problem of structure output learning can be embedded in a kernel-based framework, enabling the use of desirable kernels for a given domain, such as text or biological data.
- The main difficulty of structured output learning is to efficiently infer which output $y \in Y$ maximises the score $f(y|x)$. (Different to standard machine learning algorithms.)
- For specific models f and feature maps φ with sum-decomposability, inference can be made much faster (e.g. in linear time) with dynamic programming.
- Structured prediction is a supervised algorithm which can be advantageous compared to older unsupervised structured approaches such as HMMs.

3.3 Kernel for anomaly detection

08.06.2020

Example 3.3.1 (Hacking)
In order to detect intrusions to a system, one can extract signatures of the packet sent over the network by searching for significant patterns in malicious data. A disadvantage of this technique is that it can only detect already known attacks and is thus ineffective against attack variants.

We can use machine learning methods to identify unknown attacks by embedding the byte stream in a vector space. Assuming that malicious

byte steams somehow deviate from harmless ones, we can strive to learn a concise description of normal data.

We can use the already introduced structured kernels for the embedding. For example, we can

- construct a N-gram vector space by representing any substring of length n as a dimension. In Fig. 63, we have build a dictionary with two letter combinations. The corresponding vector entry is 1 if the two letter string appears in the message.
- construct a binary structure, which is one is the substring occurs in the message and zero else.
- construct a frequency based model such as a histogram, counting the occurrences of substrings in the message. ◇

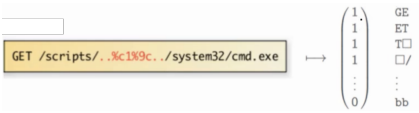


Fig. 63: **TODO: Warum ist das nicht ein binary system?**

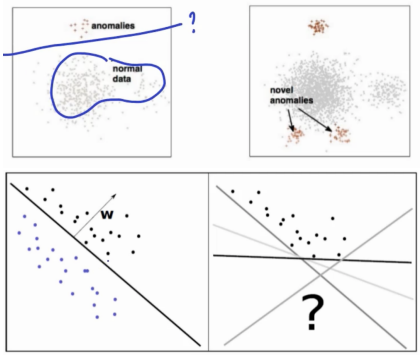


Fig. 64: One could treat this as a two class problem (with large margin) and find a decision boundary between known anomalies and normal data. This approach might be insufficient, since new anomalies must not obey this decision boundary. We thus must find a way to effectively enclosed the normal data, but it is not clear how to do that.

We present three approaches to anomaly detection.

- **Density based:** Learn a density model of the inlier data $p(x)$ and then classify a new data point as "outlier" when the probability assigned to it is low.
- **Reconstruction based:** Learn a reconstruction model of the data $x \mapsto \text{proj}_D(x)$, where D is the data manifold and then classify as outlier when the reconstruction error is too high.
- **Boundary based:** Instead of learning a surrogate method as in the two approaches above, we simply learn a separating surface between the inlier and outlier data, e.g. a hypersphere enclosing all inliers.

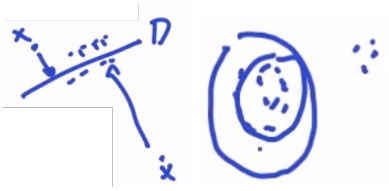


Fig. 65: **Left:** Geometric interpretation of reconstruction based methods. **Right:** The surface separating the inliers and outliers is adjusted into the direction opposite of the outliers.

Kernel density estimation (KDE)

We build a probability function

$$p(x) = \frac{1}{Z} \sum_{i=1}^{N} k(x, x_i),$$

where we typically choose $k(x, x_i) := \exp\left(-\gamma \|x - x_i\|^2\right)$, which is a Gaussian "bump" centered at x_i, and Z is a normalisation factor.

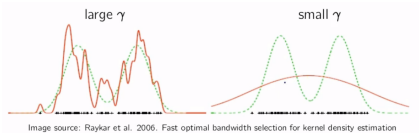


Fig. 66: **TODO**

Feature map view

Assume k induces a feature map $\varphi\colon \mathbb{R}^d \to H$. Then we have

$$p(x) = \frac{1}{Z} \sum_{i=1}^{N} k(x, x_i) = \frac{1}{Z} \sum_{i=1}^{N} \langle\, \varphi(x), \varphi(x_i) \,\rangle = \langle\, \varphi(x), w \,\rangle,$$

where $w := \frac{1}{Z} \sum_{i=1}^{N} \varphi(x_i)$ is proportional to the mean in feature space. Thus the p becomes a linear model in feature space. The function p is bounded as $\varphi(\mathbb{R}^d) = \{x \in H : \|x\| = 1\}$. **(WHYYY?)**

Kernel PCA for anomalies (uncentered)

The uncentered PCA projection in feature space can be written as

$$\varphi(x) = \underbrace{\sum_{i=1}^{a} u_i u_i^\intercal \varphi(x)}_{\text{PCA model}} + \underbrace{\sum_{i=a+1}^{h} u_i u_i^\intercal \varphi(x)}_{\text{residuals}},$$

where $(u_k)_{k=1}^a$ are the principal components. The reconstruction error / outlier score is given by the norm of the residuals:

$$o(x) = \left\| \sum_{i=a+1}^{h} u_i u_i^\intercal \varphi(x) \right\|^2 = \sum_{i=a+1}^{h} (u_i^\intercal \varphi(x))^2 \tag{15}$$

by orthonormality of the u_i.

For high dimensions, we only compute empirical principal components generated by the data, i.e. $K = U\Lambda U^\intercal$, where K is the Gram-Matrix, and U and Λ contain the eigenvectors and -values, respectively. All training points can be embedded in a feature space based on this decomposition. For a point outside the training set, we need a interpolation scheme in order to produce the kernel between new data points and old data points in the training set and the projecting them onto the principal component and then multiplying by (the scaling factor) $\lambda^{-\frac{1}{2}}$:

$$\text{proj}_i(x) := k(x, X) \cdot U_{:i} \cdot \lambda_j^{-\frac{1}{2}}.$$

We can then compute the outlier score

$$o(x) = \sum_{i=a+1}^{h} (\text{proj}_i(x))^2.$$

One can show that this coincides with (15) when x is in the training set.

Example 3.3.2 (Support vector data description (SVDD))
This boundary based model is an enclosing sphere with center c and radius $R > 0$.Points inside the sphere are considered inliers, points outside the sphere outliers. The minimum enclosing sphere can be optimised via constrained quadratic programming:

$$\min_{R,c,\xi} R^2 + \frac{1}{N\nu} \sum_{i=1}^{N} \xi_i \quad \text{s.t.} \quad \forall_{i=1}^N \|\varphi(x_i) - c\|^2 \leqslant R^2 + \xi \quad \text{and} \quad \xi_i \geqslant 0, \tag{16}$$

where the ξ_i are slack variables, one for each data point.

This is a fully unsupervised method (needs no outlier labels) and a convex optimisation problem. The quantity ν is an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors. **TODO: WHY?**

The dual problem is

$$\max_a a_i k(x_i, x_i) - \sum_{i,j=1}^{N} a_i a_j k(x_i, x_j)$$

$$\text{s.t } \forall_{i=1}^N \sum_{i=1}^{N} a_i = 1 \quad \text{and} \quad \forall_{i=1}^N a_i \in \left[0, \frac{1}{N\nu}\right]$$

residuals

reconstruction e

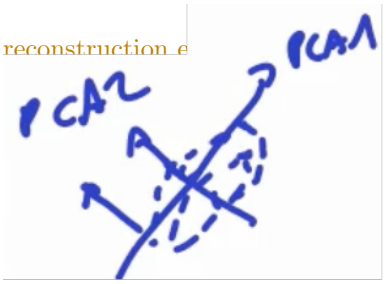


Fig. 67: Consider PCA in $\mathbb{R}^2$ with one principal component, PCA1. For an outlier the norm of the residual is the length of the projection onto PCA1.

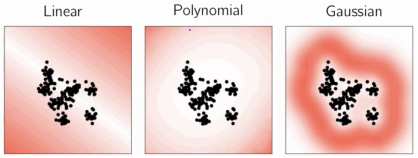


Fig. 68: For $a = 1$ (one component is reserved for the PCA model and one for the residual) we get ... for different types of kernels. For the linear model we see that the outlier function increases along the residual components, placing most of the data in the white area. But there are data points, for which the model does not work well. Furthermore there are white regions, which do not contain any inliers. The Gaussian kernel works well locally but not globally, which is clear as $e^{-x} \xrightarrow{x \to \pm\infty} 0$

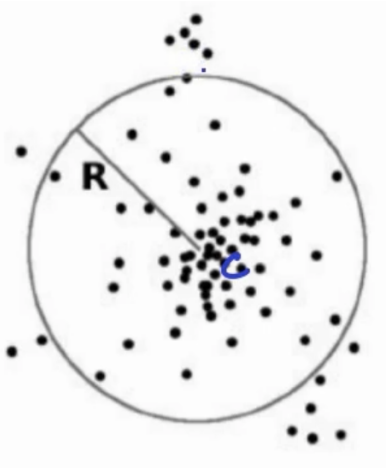


Fig. 69: TODO

Thus the dual formulation is suitable for few high dimensional data points, whereas the primal formulation works better on low dimensional data points.

If the kernel is Gaussian, only the dual problem is solvable **(???)**

One typically chooses this model if $d \gg N$ or if the feature map φ is not known. The center c can be recovered from the dual solution by $c = \sum_{i=1}^{N} a_i \varphi(x_i)$ and the radius R can be inferred from support vectors which can themselves be identified from box-constraints. (s. HW).

The outlier decision $\|\varphi(x) - c\|^2 > R^2$ can be rewritten as

$$k(x,x) - \sum_{i=1}^{N} a_i k(x, x_i) + \sum_{i,j=1}^{N} a_i a_j k(x_i, x_j) > R^2,$$

i.e. we don't need the feature map explicitly. ◇

We now turn to one-class SVM (OC-SVM), which is another variant of the problem of anomaly detection. Instead of finding an enclosing sphere, we aim to learn a hyperplane, which maximally separates the data from the origin. Here, the origin is not a specific point but a point, which does not have the specific orientation in feature space. Points closer to the origin are outliers, while points beyond the hyperplane are inliers.

The maximum separating hyperplane can be optimised via constrained quadratic programming:

$$\min_{w,\rho,\xi} \frac{1}{2}\|w\|^2 - \rho + \frac{1}{N\nu}\sum_{i=1}^{N} \xi_i \quad \text{s.t} \quad \forall_{i=1}^{N} \langle \varphi(x_i), w \rangle \geqslant \rho - \xi_i \quad \text{and} \quad \xi_i \geqslant 0.$$

The dual problem is

$$\max_{a} -\frac{1}{2}\sum_{i,j=1}^{N} a_i a_j k(x_i, x_j) \quad \text{s.t} \quad \sum_{i=1}^{N} a_i = 1 \quad \text{and} \quad \forall_{i=1}^{N} a_i \in \left[0, \frac{1}{N\nu}\right]$$

Proof. Observe that strong duality holds by Slater's condition: if there exists w, x_i, ξ, ρ with $\langle \varphi(x_i), w \rangle = \rho - \xi$, then $\langle \varphi(x_i), w \rangle < \rho - \tilde{\xi}$ for $\tilde{\xi} > \xi$. The Lagrangian is

$$L(w, \rho, \xi, a, b) := \frac{\|w\|_2^2}{2} - \rho + \sum_{i=1}^{N} \frac{1}{N\nu}\xi_i + a_i(\rho - \xi_i - \langle \varphi(x_i), w \rangle) - b_i \xi_i.$$

We have

$$\frac{\partial}{\partial w} L(w, \rho, \xi, a, b) = 0 \iff w = \sum_{i=1}^{N} a_i \varphi(x_i) \tag{17}$$

and

$$\frac{\partial}{\partial w} L(w, \rho, \xi, a, b) = 0 \iff \sum_{i=1}^{N} a_i = 1$$

and

$$\frac{\partial}{\partial \xi_n} L(w, \rho, \xi, a, b) = \frac{1}{N\nu} - a_n - b_n \overset{!}{=} 0$$

for $n \in \{1, \dots, N\}$. The dual problem thus becomes

$$\max_{a,b \geqslant 0} \frac{1}{2} \left\| \sum_{i=1}^{N} a_i \varphi(x_i) \right\|_2^2 \cancel{-\rho + \frac{1}{N\nu} \sum_{i=1}^{N} \xi_i + \rho - \sum_{i=1}^{N} a_i \xi_i} - \sum_{i=1}^{N} a_i \left\langle \varphi(x_i), \sum_{i=1}^{N} a_i \varphi(x_i) \right\rangle \cancel{- \sum_{i=1}^{N} b_i \xi_i},$$

which reduces to

$$\max_{a} -\frac{1}{2} \left\| \sum_{i=1}^{N} a_i \varphi(x_i) \right\|_2^2 = \min_{a} \frac{1}{2} \sum_{i,j=1}^{N} a_i a_j k(x_i, x_j) = \min_{a} \frac{1}{2} a^\mathsf{T} K a$$

subject to $0 \leqslant a_i \leqslant \frac{1}{N\nu}$, i.e. $\binom{-I}{I} a \preceq \frac{1}{N\nu} \binom{\mathbf{0}}{\mathbb{1}}$, where $\preceq$ denotes entry-wise inequality, and $\sum_{i=1}^{N} a_i = 1$, i.e. $a^\mathsf{T} \mathbb{1} = 1$ where $K = (k(x_i, x_j))_{i,j}$. □

The decision rule in the primal for classifying a data point as an outlier is given by $\langle \varphi(x_i), w \rangle < \rho$. Also, one can verify that for any data point x_i, whose associated dual variables satisfies the strict inequality $a_i \in \left(0, \frac{1}{N\nu}\right)$ and calling one such point a support vector x_{SV}, then the following equality holds: $\langle \varphi(x_{\mathrm{SV}}), w \rangle = \rho$. The outlier detection rule can thus, by (17) be expressed as

$$\sum_{i=1}^{N} a_i k(x, x_i) < \sum_{i=1}^{N} a_i k(x_{\mathrm{SV}}, x_i),$$

We observe that SVDD is equivalent to OC-SVM if $k(x, x) = \text{const}$ for all x, since then $\sum_{i=1}^{N} a_i k(x_i, x_i) = k(x, x) = \text{const}$ independent of α.

An example for such a kernel is the Gaussian kernel ($k(x, x) = 1$) or any other radial basis function (RBF) kernel. Geometric interpretation: all data points have unit norm in feature space: $\|\varphi(x)\|^2 = k(x, x) = 1$. The enclosing sphere and the separating hyperplane produce the same decision boundary on $\varphi(\mathbb{R}^d)$.

The boundary approach is interesting as one has flexibility in the modelling task: One can generalise SVDD by including labels in a SVM fashion and then make them subject to a different constraint, for example that they should be outside the hypersphere. This can affect the hypersphere compared to what we would learn with only inlier data.

The semi-supervised (unlabeled and labeled data) setting can implemented by modifying (16), the extended SVDD is

$$\begin{aligned} \min_{R,\gamma,c,\xi} \; & R^2 - \kappa\gamma + \nu_u \sum_{i=1}^{n} \xi_i + \eta_I \sum_{j=n+1}^{n+m} \xi_j \\ \text{s.t. } & \forall_{n=1}^{N} \|\varphi(x_i) - c\|^2 \leqslant R^2 + \xi_i, \quad \xi_i \geqslant 0, \\ & \forall_{j=n+1}^{n+m} y_j \left(\|\varphi(x_j) - c\|^2 - R^2 \right) \leqslant -\gamma + \xi_j, \quad \xi_j \geqslant 0, \end{aligned}$$

where γ is the confidence margin. Unfortunately, this problem is not convex, but there exists a convex relaxation.

For each kernel based method, there also a deep counterpart:

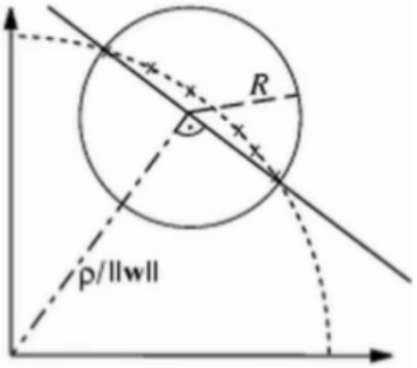


Fig. 70: **TODO**

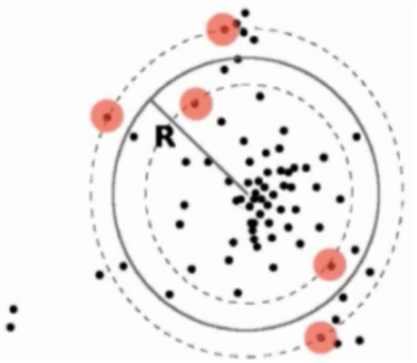


Fig. 71: **TODO**

-based /	Kernel	Deep
Density	KDE	DBMs, Hierchical latent
Reconstruction	KPCA	Autoencoder
Boundary	OC-SVM	GAN, deep OC-SVM

and many more (isolation forest, local outlier factor).

Beyond Prediction: Explaining Anomalies

Sometimes, it is important to not only detect that a point is anomalous, but also to understand why it has been classified as such to very that the detection is justified.

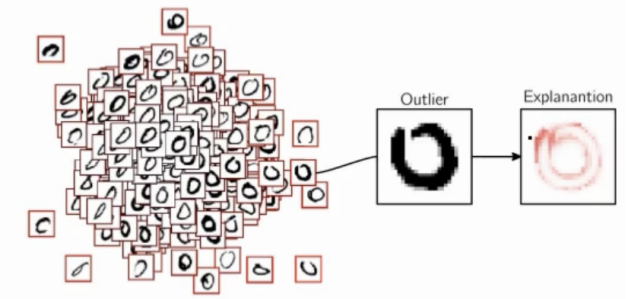


Fig. 72: The explanation highlights which pixels are responsible for the anomaly detection.

In order to explain KDE and OC-SVM, we note that both models are of the type

$$f(x) = \sum_i a_i \exp\left(-\gamma \|x - x_i\|^2\right),$$

which is closed to an inlier function because [inaudible].

We can convert it into an outlier function by applying a decreasing function to it:

$$\begin{aligned} o(x) &= -\frac{1}{\gamma}\log(f(x)) = -\frac{1}{\gamma}\log(\sum_i \exp\left(-\gamma\|x - x_i\|^2 - \log(a_i)\right) \\ &= \min_i{}^{\gamma}\{\|x - x_i\|^2 - \log(a_i)\}, \end{aligned}$$

i.e. a soft minimum over the squared distances, which can be interpreted as the minimum distance of x to an inlier. This outlier score can there be redistributed in two steps

- min-take-most in the pooling layer ($\arg\min_i^{\gamma}\{\cdot\}$) (in this step, the x_i, which attains the minimum in the outlier score, is identified),
- directional redistribution in the distance layer $\frac{(x-x_i)^2}{\|x-x_i\|^2}$ (given use attribution on input features of x_i).

Fig. 73: The lines represent kind of level lines, to illustrated the outlier function.

4 Deep Learning

4.1 Neural Networks for Structured Data

Neural networks recapitulation

With neural networks we want to solve nonlinear problems, we show three approaches to tackle such problems.

Example 4.1.1 (Feature Engineering)
In the most basic approach one extracts features from the input, which one believes to be relevant for the task. If one has extracted good features, on can train a linear classifier on top of those features and get good results. Typically, features are designed to incorporate knowledge.
A disadvantage of this approach is that for every new task one needs to code new features, so this approach fails for tasks, where we don't know the features a priori. ◇

Example 4.1.2 (Kernels / Expansions)
The polynomial kernel $k(x, y) := (1 + \langle x, y \rangle)^d$ induces the feature map

$$\varphi(x) := \left[1, \sqrt{d}(x_i)_i, \sqrt{\frac{d(d-1)}{2}}(x_i x_j)_{ij}, \dots\right]^\mathsf{T},$$

from which we can represent the task linearly (if d is large enough). A disadvantage of this automatic feature expansions approach is that one never knows if the expansion will produce features that are relevant for the task of if there will be a lot of redundancies and features modelling task-unrelated features. ◇

Example 4.1.3 (Neural networks)
Neural networks are inspired by the brain, which has the ability to learn a problem representation using a large but finite number of neurons. A(n) (artificial) neural network is a machine learning model that mimics this capability.

A visual stimulus can be a input x for the brain, which produces a neural response, which can be thought of as the feature map $\varphi(x)$. Something unique about this neural network is that the problem representation can be learned: by inputting several stimuli over a certain period of time, the neural response will change: it will increasingly well represent what has been received as inputs. This feature is called "plasticity".

We simplify the biological neuron to only retain the essential components for learning: nonlinearity and plasticity. The nonlinearity can be implemented by some activation function g and the plasticity is implemented by the parameters w_{ij} and b_j, which are learned from the data.

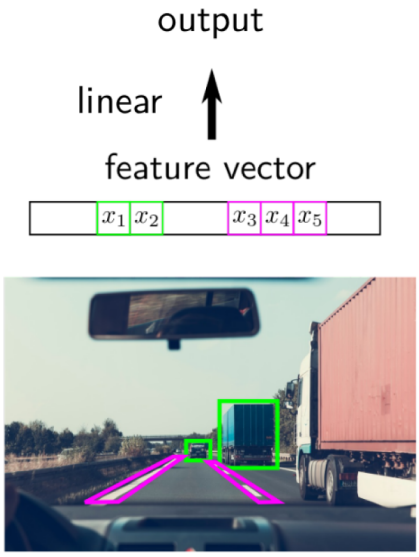


Fig. 74: To build a assistance to drive on likes to detect the lanes and the other vehicles. Based on those features, the classifier might take a decision such as turning left or right. (The vector should rather look like $\dots, x_3, x_4, \dots, x_7, x_8, x_9, \dots$.)

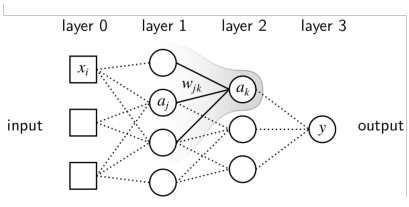


Fig. 75: A simple three-layer network

The forwards pass of the neural network in Fig. 75 is

$$z_j = \sum_i x_i w_{ij} + b_j \qquad a_j = g(z_j) \qquad \text{(layer 1)}$$

$$z_k = \sum_j x_i w_{jk} + b_k \qquad a_j = g(z_k) \qquad \text{(layer 2)}$$

$$y = \sum_k a_k v_k + c, \qquad \text{(layer 3)}$$

where i represents the inputs in layer 0, j the neurons in layer 1 and k the neurons in layer 2 and e.g. w_{ij} is the weight of the edge connecting x_i and a_j.

The error gradient can be computed efficiently with error backpropagation, which reuses intermediate computation in the graph to be able to compute the gradient in linear time of the number of neurons (instead of exponential time) Consider an error function E depending on the output y and some targets (e.g. ground truth) t. We then have

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \underbrace{\sum_k \frac{\partial z_k}{\partial z_j} \underbrace{\frac{\partial y}{\partial z_k}\frac{\partial E}{\partial y}}_{\delta_k}}_{\delta_j}. \tag{18}$$

The last terms only depend on z_k and thus be stored in the variable δ_k and need not be recomputed, and similarly for δ_j. ◇

Example 4.1.4 (Shared parameters (HA 7-1)) Let x_1, x_2 be two observed variables. Consider the two-layer neural network that takes these two variables as input and builds the prediction y by computing iteratively

$$z_3 = x_1 w_{13}, \quad z_4 = x_2 w_{24}, \quad a_3 = \frac{1}{2} z_3^2, \quad a_4 = \frac{1}{2} z_4^2, \quad y = a_3 + a_4$$

Consider the loss function $\ell(y,t) := \frac{1}{2}(y-t)^2$, where t is the target variable that the neural network learns to approximate. Using the rules for backpropagation (cf. (18)) yields

$$\frac{\partial \ell(y,t)}{\partial w_{13}} = \frac{\partial \ell(y,t)}{\partial y}\frac{\partial y}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = (y-t) z_3 x_1$$

and, similarly, $\frac{\partial \ell(y,t)}{\partial w_{24}} = (y-t) z_4 x_2$. Since the variables y and z_i are computed in the forward pass, they are available and one need not express them in terms of the input variables and the weights.

Assume that the weights cannot be adapted freely, but are a function of the same shared parameter v:

$$w_{13} = \log(1+\exp(v)), \qquad w_{24} = -\log(1+\exp(v)).$$

Assuming, we have computed $\frac{\partial \ell(y,t)}{\partial w_{13}}$ and $\frac{\partial \ell(y,t)}{\partial w_{24}}$, we obtain

$$\frac{\partial \ell}{\partial v} = \frac{\partial \ell(y,t)}{\partial w_{13}}\frac{\partial w_{13}}{\partial v} + \frac{\partial \ell(y,t)}{\partial w_{24}}\frac{\partial w_{24}}{\partial v} = \frac{(y-t) z_3 x_1}{1+e^{-v}} + \frac{(y-t) z_4 x_1}{1+e^{v}}. \quad \diamond$$

Remark 4.1.5 (Machine Learning Desiderata)

- **Universality.** The method can be applied to any task, also on such where we do not know what the features should be a priori.
- **Compactness.** We don't want the model to require exponentially neurons / features to represent a certain task.
- **Convexity.** We want our algorithm to converge to the optimum.

◊

The feature engineering approach is compact as we just have to select a few features for the task. The kernel expansion approach might produce very large expansions before being able to extract task relevant features, so one ends up with exponentially many features to deal with. Fig. 76 shows that that neural networks are universal: think of the input neurons as extracting a direction in the input space. If one combines several of the such obtained tanh functions (we choose $g \equiv \tanh$), we obtain a "bump" with sufficiently many neurons. One more layer now combines many different bumps and many different locations, which gives the power to approximate any function. This can be thought of as a kernel map which produces an RBF function at a certain location. If these kernel maps are universal approximators, so are the neural networks.

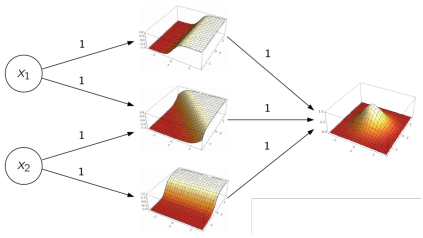


Fig. 76: Constructive "proof" of universality of neural networks.

The feature engineering approach is convex as we just have to train the top layer classifier **(???)**. The kernel approach is also convex as they are a way of representing a linear model in a induced feature map. We will show that neural networks are not convex, but they can still be optimised with heuristic effort.

Approach	Universal	Compact	Convex
Feature engineering	No	Yes	Yes
Kernels / Expansions	Yes	No	Yes
Neural networks	Yes	Yes	No

Table 2: Machine learning desiderata and the three approaches.

Compactness of neural networks

Consider approximating the parity function

$$f\colon \{0,1\}^d \to \{0,1\},\ x \mapsto \text{parity}(x_1, \ldots, x_d),$$

which is one if and only if the number of ones in x is odd and zero else. Plotting $\{0,1\}^d$ one a hypercube and colouring the vertices according to f, one sees that neighbouring vertices always have the opposite colour.

A naive approach using one neuron / basis function to handle each corner of the hypercube yields 2^d neurons, whilst building the composition $(f(x) = x_1 \vee x_2 \vee \ldots \vee x_d)$ progressively yields only $4d$ neurons: building a bit parity between the first two dimension can be realised with four neurons. Adding one dimension takes four neurons again. This architecture is way deeper than the naive approach, but avoids the curse of dimensionality.

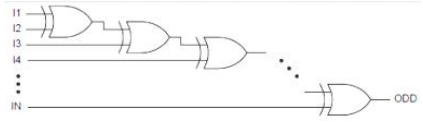


Fig. 77: Progressive exclusive-or composition. [dai]

More practically, one task of interest is image recognition. Solving this problem with random basis functions, they would typically look like random images and one might need a lot of them to be able to extract good feature extraction. The idea is that if one can train the neurons, each of the neurons might convergence to useful and very clean filter (edge / color / high frequency texture detectors). **Slide 12 is very confusing.**

Remark 4.1.6 (Optimisation of neural networks)
Neural networks are non-convex: Even the simplest two-layer networks $\varphi(x, \theta) := \theta_1\theta_2 x$ is non-convex in θ. Thus many hyperparameters (e.g. initialisation, learning rate, etc.) can affect the learning outcome.

Multiple layers can cause pathological curvature, i.e. the gradient vanishing along certain directions of the parameter space. The optimiser might then get stuck on large plateaus.

With heuristics on the neural network design (e.g. choice of layers and nonlinearities (ReLU works well, sigmoid doesn't) and optimisation (whether one should use momentum, where one should reparametrise the network) it is however possible to train neural networks efficiently. ◊

Neural networks for structured data

Molecules, volumetric data, parsing trees, images or documents (e.g. historical tables), DNA sequences and general graphs are all high-level structures we want to classify.

Example 4.1.7 (Neocognitron (1980))
For image recognition, the Neocognitron was proposed 40 years ago in [Fuk80]. It very much resembles convolutional neural networks we still use nowadays. The Neococognitron is an architecture for representing images, where the produced representation is not affected by a translation of the input image: If one moves elements of the image by some amount, they will also move by that amount in the convolutional layers. In the pooling layers takes a receptive field and aggregates all what one sees in a specific location. This process can be alternated; progressively, the relevant information persists but the spatial information becomes more and more coarse until it becomes completely invariant to the exact positing. We say the Neocognitron consists of an alternation of "simple cells" (convolutions) and "complex cells" (pooling), which progressively build the desired representation. ◊

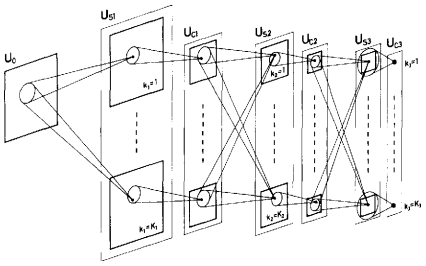


Fig. 78: Schematic diagram illustrating the interconnections between layers in the neocognitron. [Fuk80]

Convolutional neural networks

Then came the convolutional neural network (CNN) (1989, 2012) [L$^+$89].

Here we not only have this alternation of convolution and pooling layers, but they will be provided with an algorithm to train their parameters using the backpropagation algorithm which was introduced some years earlier. This architecture is, up to very minor changes (ReLU instead of tanh nonlinearities), the state-of-the-art in many image recognition tasks, which became apparent, when the ILSVRC (ImageNet Large Scale

Visual Recognition Challenge) 2012 was won by a wide margin by CNNs. CNNs are resource-hungry for training data, typically they have to be trained on GPUs. One advantage of CNNs is that there are a lot of repositories with pretrained CNNs. The idea is that one can just cut the last classification layer and use it as a feature map and just train a linear or shallow classifier on the task. There are a lot of applications, where generic neural networks for images have been transferred to tasks in medical applications.

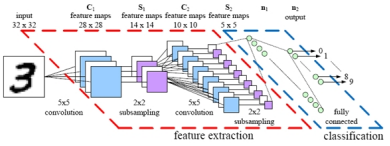


Fig. 79: Architecture of a CNN. [NL]

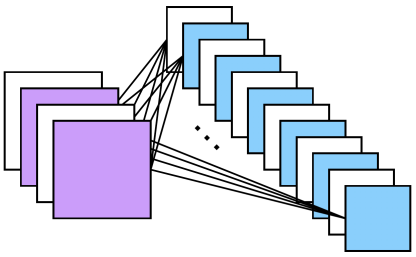


Fig. 80: The convolution layer.

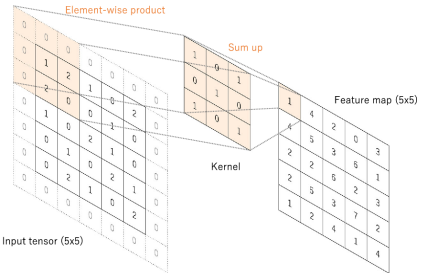


Fig. 81: A convolution operation with zero padding so as to retain in-plane dimensions. [YNDT18]

A convolutional layer takes in different input tensors that are response maps to the input image, e.g. one response to horizontal / vertical edges or color blobs, which are of the same dimensions as the input image. One creates a layer of convolution, where each output response map will be the weighted sum of the convolution of the input response map with a convolution kernel:

$$\forall_j a^{(j)} = g\left(\sum_i w^{ij} \star x^{(i)}\right).$$

Typically, one doesn't use convolution but cross-correlation, which is the same except some flipping of the convolution kernel: $z^{(ij)} = w^{(ij)} * x^{(i)}$ The input tensor is an array of numbers and the convolution kernel is w^{ij}. One applies the kernel to the first input patch (orange) and the slide it over the matrix, yielding a new response map which contains all the dot products between the input patches and the convolution kernel.

An advantage of cross-correlation is that one can backpropagate the gradient in an efficient way:

$$z_t = [w \star x]_t = \sum_{s\in\mathbb{Z}} w_s \cdot x_{t+s}.$$

Consider E to be an error function depending on z. By the chain rule the backward pass is

$$\frac{\partial E}{\partial x_u} = \sum_{t\in\mathbb{Z}} \frac{\partial E}{\partial z_t}\frac{\partial z_t}{\partial x_u} = \sum_{t\in\mathbb{Z}} \frac{\partial E}{\partial z_t} w_{u-t} = \left[\frac{\partial E}{\partial z} * w\right]_u$$

and the parameter gradient

$$\frac{\partial E}{\partial w_u} = \sum_{t\in\mathbb{Z}} \frac{\partial E}{\partial z_t}\frac{\partial z_t}{\partial w_u} = \sum_{t\in\mathbb{Z}} \frac{\partial E}{\partial z_t} x_{u+t} = \left[\frac{\partial E}{\partial z} \star w\right]_u$$

Consider the activation function g to be $a_t^{(\ell)} := \max(0, z_t^{(\ell)})$, where the forward pass is

$$z_t^{(\ell)} = \sum_{k=1}^{K} \left[x^{(k)} * w^{(k,\ell)}\right]_t = \sum_{k=1}^{K}\sum_{s\in\mathbb{Z}} x_s^{(k)} w_{t-s}^{(k,\ell)},$$

where K is the number of input signals to the convolutional layer. The convolutional filters $w = ((w^{(k,\ell)})_{k=1}^{K})_{\ell=1}^{L}$ have to be learned from the data. After passing the data through the convolutional layer, the neural networks output is given by some function $f(a)$. To learn the model, the parameter gradients need to be computed. Assuming one knows $\frac{\partial f}{\partial a}$, we

can express $\frac{\partial f}{\partial w}$ via the chain rule:

$$\begin{aligned}\left[\frac{\partial f}{\partial w^{(k,\ell)}}\right]_u &= \sum_{t\in\mathbb{Z}} \frac{\partial f}{\partial a_t^{(\ell)}} \cdot \mathbb{1}(z_t^{(\ell)} > 0) \cdot \frac{\partial z_t^{(\ell)}}{\partial w_u^{(k,\ell)}} \\ &= \sum_{t\in\mathbb{Z}} \frac{\partial f}{\partial a_t^{(\ell)}} \cdot \mathbb{1}(z_t^{(\ell)} > 0) \cdot \frac{\partial}{\partial w_u^{(k,\ell)}} \sum_{s\in\mathbb{Z}} x_s^{(k)} w_{t-s}^{(k,\ell)} \\ &= \sum_{t\in\mathbb{Z}} \frac{\partial f}{\partial a_t^{(\ell)}} \cdot \mathbb{1}(z_t^{(\ell)} > 0) \cdot x_{t-u}^{(k)}.\end{aligned}$$

We have seen that CNNs are

- universal: the CNN model is (this is empirically verified) able to recognise most known visual objects (ImageNet classes, handwritten digits, traffic signs, spectrograms (speech), volumetric data from the brain, other time series, etc.) Because the convolution network is not relying on a specific filter but instead a general architecture which consists of building the features progressively, it does not limit itself to a specific set of classes.
- compact: the representation remains finite-dimensional (as in not exponentially many neurons) at each layer. The spatial resolution is progressively traded for semantic resolution: One progressively looses spatial resolution, the feature map becomes less and less resolved until they are not spatially resolved at all: they correspond to a single scalar. On the other hand one increases the semantic content of the representation: the number of filters increases as their spatially resolution decreases. This makes sense as the more one has detected high level concepts, the less one has to keep track of their position in order to be able to relate them. We arrive at a very high dimensional representation with no spatial resolution from which we can build a classifier.
- nonconvex. However, it still converges to good solutions with a careful initialisation and choice of optimisation parameters.

Remark 4.1.8 (Adding CNNs to high-level structure)
CNNs need many labels to preform well. In new problems, one may have less labels because it requires experts or simply because there is not enough interest in the task to be able to assign enough resources to it. In that case there are two possible approaches: the first one is to take a pretrained network and retrain the top layer, which might work well if the task is related. For other tasks with very specific composition in the input image, this might not be optimal. In this case one can train the CNN on local patches and augment it with a high-level structure, which is making use to prior knowledge (e.g. known compositions and invariance). One can hardcode this in the top layer of the CNN and get a classifier solving the task. ♢

Example 4.1.9 (Bigram network [EBK+20])
The Bigram network (developed at TUB, too) aims to detect if two tables from historical textbooks are similar. If one inputs such a table into a state-of-the-art image recognition CNN, it would probably not detect what the digits are and how they are related and not be able to ignore

the perturbing factors that are the lines of the table. It is also not clear, what a similarity measure for tables should be, if it focuses on the overall cell structure or the content within.

For the task of matching tables based on their numeric content, one feeds the table to a standard digit-recognising CNN. Instead of building a single output, the CNN outputs are response maps for every digit. The idea is that one can hardcode some layers (high-level structure) by using min-pooling to compose two adjacent digits:

$$a_{jk}^{(\tau)}(x) := \min(a_j(x), \tau(a_k(x))),$$

where a_j is the activation map for digit j, where τ is some small translation operation, which account for a variety of shifts (there are tables where the digits are more condensed than in others). Instead of detecting digits, one can now detect two-digit numbers. We build a local shift invariance by having a max-pooling over these shifts:

$$\max_{\tau} a_{jk}^{(\tau)}(x).$$

The min can be understood as a logical soft and, which tests whether we have an activation both in a_j and a_k. If one of them is missing, we have $\min(\cdot, 0) = 0$. The max can be interpreted as a soft logical or.

Once we have done the local shift invariance, we can build translation invariance:

$$\varphi_{jk}(x) := \|a_{jk}(x)\|_1, \qquad \diamond$$

which yields a histogram of all digits in the table, which can be used as a representation to compare tables.

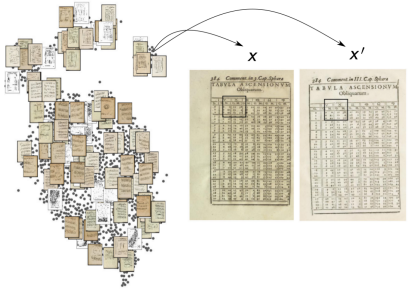


Fig. 83: Based on their similarity, the historical tables can be mapped into an embedded space using e.g. t-SNE. Points that are mapped to the same location, can indeed be verified to have a similar numerical content, they would often correspond to the same table taken from different books.

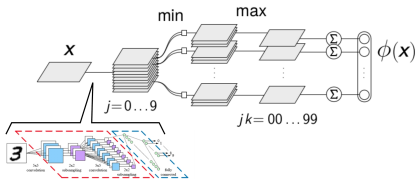


Fig. 82: TODO

Text classification

In the context of image classification, images are seen as arrays of pixels and every pixel is given by its RGB components $x_p \in [0,1]^3$. Therefore, they can be readily given as input to a neural network.

On the other hand, texts are collections of words. Words are abstract symbols that do not possess a numerical representation. Instead they

need to be embedded into one such representation so that they can be given as an input to a neural network. There are various ways of vector embedding

- One-hot encoding: Represent a word as a one-sparse vector:

$$e[w] := (\mathbb{1}(w =' a'), \mathbb{1}(w =' able'), \mathbb{1}(w =' about')) \in \mathbb{R}^{\#\text{words}},$$

  i.e. $e['able'] = (0, 1, 0, \ldots)$. This approach is very simple to setup and doesn't loose any information and any pair or words can be linearly separated, but this is a very high dimensional representation, which can be very inefficient by the curse of dimensionality

- kPCA embedding: One relies on a structured kernel (cf. lecture 4) k, which measures similarity between words (e.g. co-occurrence of lexical similarity). The embedding is obtained by diagonalising the Gram matrix $K = (k(x_i, x_j))_{i,j} = U\Lambda U^\intercal$ and defining

$$e[w] := U_{w,:d} \odot (\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d}) \in \mathbb{R}^d .$$

  Typically, this will retain most of the variance: one will get useful principal components, which hopefully capture all of the task subspace. We want d to be in the order of a few hundreds and not many thousands.

- Learned embedding (e.g. Word2Vec): Start with a random embedding $e[w] \in \mathbb{R}^d$ and learn the embedding on the supervised task (e.g. text classification) directly or on some related unsupervised task (checking if the embedding of two similar words is similar) E via the update

$$e[w] \leftarrow e[w] - \gamma \frac{\partial E}{\partial e[w]}$$

  for some learning rate $\gamma > 0$.

Once we have such an embedding for words, we can build a neural network architecture for a sentence. One approach is to use recursive neural networks (RecNNs), which makes use of a parsing tree of the sentence, which can always be extracted based on some rules (cf. Stanford parser), if the sentence if grammatically correct). If one has a binary parsing tree, one can build a function, which is applied recursively from the leaves at every node to finally arrive at the complete sentence. From the root, something can be predicted about the sentence.

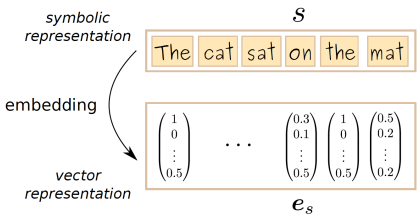
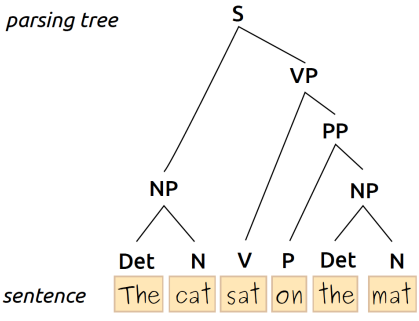
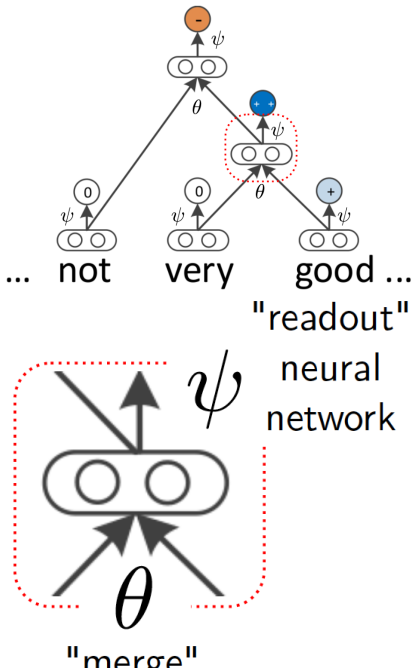


Fig. 84: A vector embedding on a sentence.





Example 4.1.10 (RecNNs for sentiment analysis [SPW+13]) Sentiment analysis aims to find out whether a sentiment is positive or negative. The "merge" neural network takes two words and merges them into one word. The "readout" neural network, which takes the output of the merge function and converts it to a scalar, represents the actual sentiment prediction. Ideally, one would like to use this function only at the root of the tree, but since "readout" always maps to the same domain, we can also apply this function anywhere in the tree. ◇

Graph neural networks (GNNs)

GNNs are networks specialised for classifying graphs, which are a very general data representation including pixel lattices, trees or sequences as special cases. GNNs receive a graph as input and implement, at each layer, a propagation step along its edges. The parameters of that propagation can be learned.

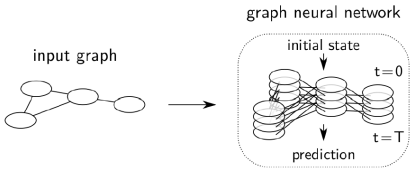


Fig. 87: The input not layer zero, it is the graph, which is a initial state. At each layer, the GNN preforms at diffusion step in the graph until one achieves a prediction. Thus the input graph is at every layer of the GNN. [SEL+20]

Example 4.1.11 (GNN from [KW16])
Consider a graph with adjacency matrix A and associated normalised Laplacian $\Lambda_t := D^{-0.5}AD^{-0.5}$, where D is a diagonal matrix containing the degrees of the nodes in the graph. If one multiplies the Laplacian by itself several times, it does not produce [inaudible] data only for high-degree nodes but it will implement some diffusion process in that graph. Let H_0 be some initial state (cf. $t = 0$ in Fig. 87) and W a set of trainable parameters of GNN, which we learn from the data.

The t-th layer of a GNN consists of two matrix multiplications follows by a nonlinearity g:

$$H_t := g(\Lambda_t Ht - 1W_t).$$

Multiplication of H_{t-1} by Λ_t performs a pooling of messages coming from neighbouring nodes, making the states propagate along the edge of the graph. Multiplication of H_{t-1} by W_t (which can be seen as a rotation in the dimension of initial state) extracts features that are relevant for the prediction task.

Progressively, one hops that information circulates along the graph to e.g. detect graph patterns like highly connected areas. This will become more apparent after several layers of representation. ◇

Example 4.1.12 (Application of GNNs to molecules)
A molecule can be expressed as a graph. The GNN exchanges information between the different atoms to identify interesting structures and reaches a prediction. This accelerates quantum chemical computation by bypassing expensive DFT simulations, which give us the true quantum chemical properties of the molecule. When comparing many molecules, the slow DFT is infeasible. ◇

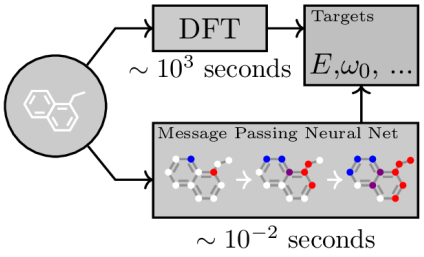


Fig. 88: A Message Passing Neural Network predicts quantum properties of an organic molecule by modelling a computationally expensive DFT calculation. [GSR+17]

Example 4.1.13 (Application of GNNs to meshes)
Recognising objects not from images but from meshes, where instead of an image we have a 3D-model, can be solved with GNNs, too. We define a graph on the mesh and each diffusion step takes the nodes of that graph which are connected to nodes of interest. In every step the dimensionality of the state increases. Finally, a fully connected layer classifies the mesh. ◇

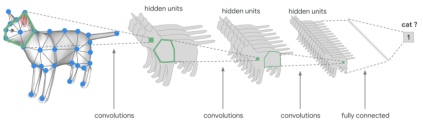


Fig. 89: Source?

Example 4.1.14 (Application of GNNs to)
GNNs can also be used in the context of semi-supervised learning. We have a graph, which is not a single data point, but a whole data set. Similar data points are connected by an edge. Several layers of the NN "let the points communicate". Only labels for some data points exists. Due to the diffusion steps, the classes of points that are unlabelled can be predicted with a readout function. ◇

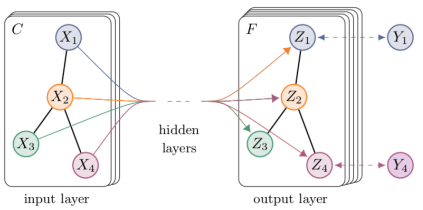


Fig. 90: [KW16]

Remark 4.1.15 (CNNs as a special case of GNNs)
GNNs are general and incorporate CNNs as a special case, where the graph is a two-dimensional lattice. Although, it would not be a good idea to implement CNNs directly as GNNs because CNNs have an efficient way of structuring computations. In the image, the red node has a receptive field of size 3 x 3 and the convolution takes the neighbours and computes a center point. ◊

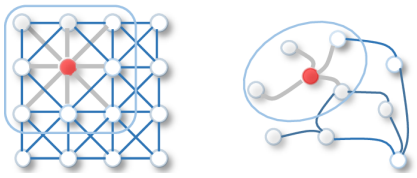


Fig. 91: **Left:** 2D convolution. Analogous to a graph, each pixel in an image is taken as a node where the neighbours are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbours. **Right:** Graph convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbours. [WPC+20]

In summary,

- neural networks can learn compact representations for any relevant tasks, because it does not extract hardcoded features. This joint capability comes at the cost of a more difficult (nonconvex) optimisation problem.
- Structured networks are generalisation of standard networks that allow handling of various real world data, e.g. images by CNNs, text (via an embedding) by RecNNs and graphs by GNNs.
- CNNs efficiently extract image representations by progressively expanding the semantic content while compress inf the pixel-wise information.
- GNNs generalised NNs to any graph-structured input, including trees and pixel lattices.

4.2 Structured Prediction

Motivation (Forward vs. Inverse Problem)
In Fig. 92 one can see a typical machine learning problem, where one has an input x and an unknown function f producing the target t via $t = f(x) + \text{noise}$. We can approximate f with standard neural networks, as one can predict the expected target given the input, which is a deterministic function. If one wants to solve an inverse problem e.g. predicting the input from only the output, one can run into trouble as the function f mapping the input to the target is not invertible. Thus for given input, there are several possible outputs. The idea for tackling an inverse problem thus is to view the task of prediction as that of learning a conditional probability model $p_\theta(t|x)$: for given x we look at all the possible values of t and build a probability distribution of $t|x$. We then minimise the objective $\min_\theta D\left(p(t|x)\|p_\theta(t|x)\right)$, where D is some divergence measure between the two distributions such as the KL-divergence of the Wasserstein distance. Here, $p(t|x)$ is the true target distribution, which will simply be an empirical estimate of the target given x.

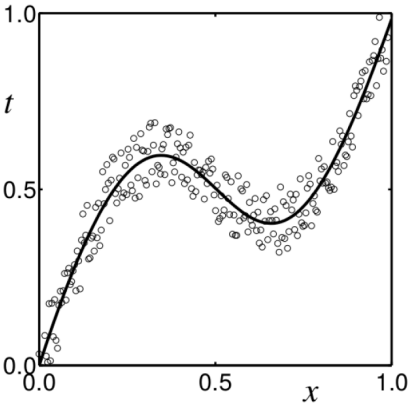


Fig. 92: **todo: citation**

Mixture Density Networks (MDNs)

MDNs are neural networks specialised for predicting conditional probability densities, which typically are (Gaussian) mixture models. The MDNs will not predict the output directly but instead the parameters of the distribution.

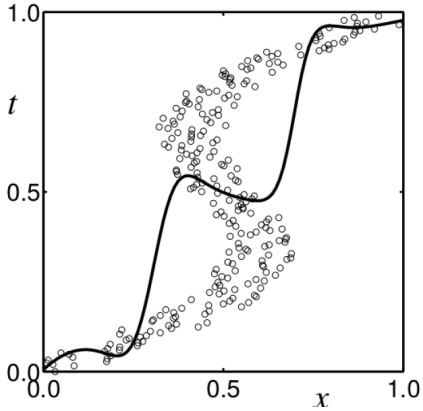


Fig. 93: Taking the mean of multiple outputs may result in a bad prediction: the mean point is at a point where there are no data points.**todo: citation**

First we model the output as a Gaussian mixture:

$$p(t|x) = \sum_{i=1}^{m} a_i(x)\varphi_i(t|x),$$

where

$$\varphi_i(t|x) := \frac{1}{(2\pi)^{\frac{c}{2}}\sigma_i(x)^c} \exp\left(-\frac{\|t-\mu_i(x)\|^2}{2\sigma_i(x)^2}\right)$$

are the mixture elements. The parameters of the mixture are the output of the network. One then optimises Gaussian mixture's likelihood

$$E^q := -\ln\left(\sum_{i=1}^{m} a_i(x^q)\varphi_i(t^q|x^q)\right),$$

which is he contribution to the error function of one data point q. Using Bayes' theorem, the posterior distribution is

$$\pi_i(x,t) := \frac{\alpha_i\varphi_i}{\sum_{j=1}^{m}\alpha_j\varphi_j}.$$

Simple calculations show that $\frac{\partial E^q}{\partial a_i} = -\frac{\pi_i}{a_i}$ and $\frac{\partial E^q}{\partial \mu_{ik}} = \pi_i\frac{\mu_{ik}-t_k}{\sigma_i^2}$. With theses gradients, we can backpropagate to update the parameters of the network.

The parameters are subject to constraints, for example, a, the probability of generating a component of the mixture is nonnegative and all a sum up to 1. We thus reparametrise

$$a_i = \frac{\exp(z_i^a)}{\sum_{j=1}^{M}\exp(z_j^a)}$$

with a softmax function to ensure these constraints hold true. We then get

$$\frac{\partial E^q}{\partial z_i^a} = \sum_{j=1}^{m}\frac{\partial E^q}{\partial a_j}\frac{\partial a_j}{\partial z_i^a} = \sum_{j=1}^{m} -\frac{\pi_j}{a_j}\left(\delta_{ij}a_i - a_i a_j\right) = a_i - \pi_i.$$

There are no constraints on μ, so we write $\mu_{ik} = z_{ik}^{\mu}$. As σ_i has to be positive, we reparametrise as $\sigma_i = \exp(z_i^{\sigma})$. We have build the parameter vector z from Fig. 95.

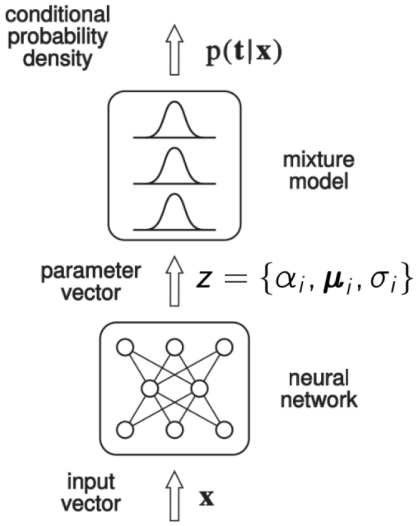


Fig. 95: The output of the network are the parameters $z = (a_i, \mu_i, \sigma_i)$. If we have a mixture of three components, we have three values of a, three vectors μ and three positve scalars σ. **TODO citation 2 from slides**

Example 4.2.1 (MDNs for inferring planet composition)
A data set of artificial planets is build form physics-based simulations. We want to predict certain hidden properties (composition) of a planet from a limit number of observed variables (mass/radius/k_2). But for certain set of parameters, there might be several composition that correspond to those parameters. We use a MDN to handle this multi modality.

The MDN model can first be verified on planets for which the interior structure is known. **TODO: can find plot form slide 7** he MDN model can then be used to infer the com,position of less known exoplanets from a limited number of measurements. ◇

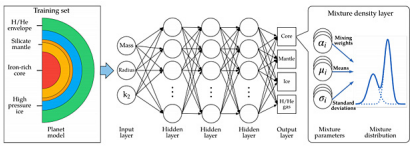


Fig. 96: [BPT+20]

Boltzmann machines

Gaussian mixture models (GMMs) capture local regions of high density but are not efficient for distributions that are shaped by global effects.

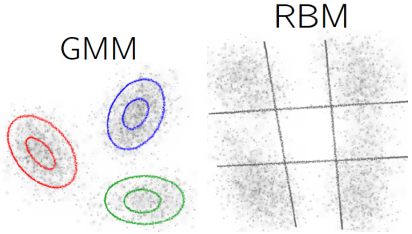


Fig. 97: One can think of the Boltzmann machine as implementing (linear) decision boundaries in the input space and attributing high probability to points that are on one side and low probability for points on the other side.

Boltzmann machines (BMs) (sometimes called product of experts) describe the probability function as a product of factors. Each factor captures a global effect in the distribution.

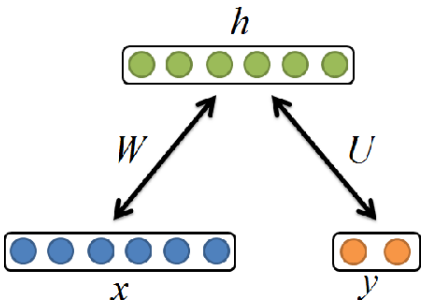


Fig. 98: [JyHL+14]

Example 4.2.2 (Conditional RBM)
A conditional restricted Boltzmann machine (CRBM) is a system of binary variables composed of an input $x \in \{0,1\}^d$, an output $y \in \{0,1\}^c$ (t in the previous subsection) and a latent state $h \in \{0,1\}^K$. It is governed by an energy function

$$E\colon \{0,1\}^{d+c+K},\ (x,h,y) \mapsto -x^\mathsf{T} W h - y^\mathsf{T} U h$$

and associates to each joint configuration the probability

$$p(x,h,y) := Z^{-1} \exp\left(-E(x,h,y)\right),$$

where Z is a normalisation term (chosen such that $\sum_{x,h,y} p(x,h,y) = 1$).

We can calculate

$$p(h_k = 1|x,y) = s(x^\mathsf{T} W_{:,k} + y^\mathsf{T} U_{:,k}) \quad \text{and} \quad p(y_j = 1|h,x) = s(U_{j,:}^\mathsf{T} h),$$

where $s(t) := \frac{1}{1+e^{-t}}$ is the sigmoid function. Furthermore, marginalisation can easily be achieved: $p(x,y) = Z^{-1}\exp(-F(x,y))$, where

$$F(x,y) := -\sum_{k=1}^{K} \log(1 + \exp(W_{:,k}^\mathsf{T} x + U_{:,k}^\mathsf{T} y))$$

is called the free energy (think of it as energy that has been freed from the latent variable) and can be interpreted as a two-layer neural network (activation function is $\log(1+\exp(x))$!). The lower the free energy, the more likely the joint configuration (x,y). Hence, structured prediction can be performed as

$$y|x = \arg\min_y F(x,y).$$

We see that the lower the energy of a configuration, the more like it is. This is different from the scores we used in the context of structured output for kernels. This is mainly because the system are inspired by physics, where probability relates to negative energy: high probability relates to low energy, as physical system tend to be in a state of low energy in most cases, i.e. with a high probability. ◇

Example 4.2.3 (Training an CRBM)
A common training procedure is to maximise the data likelihood $\ell := \frac{1}{N}\sum_{n=1}^{N} \log(p(x^n, y^n)$, where $(x^n)_{n=1}^N$ and $(y^n)_{n=1}^N$ are the data points. Its gradient is given by

$$\begin{aligned}
\frac{\partial \ell}{\partial W_{ik}} &= \mathbb{E}_{\hat{p}(x,y)}[x_i \cdot s(W_{:k}^\mathsf{T} x)] - \mathbb{E}_{p(x,y)}[x_i \cdot s(W_{:k}^\mathsf{T} x)] \\
\frac{\partial \ell}{\partial U_{jk}} &= \mathbb{E}_{\hat{p}(x,y)}[y_i \cdot s(U_{:k}^\mathsf{T} y)] - \mathbb{E}_{p(x,y)}[y_i \cdot s(U_{:k}^\mathsf{T} x)],
\end{aligned}$$

where $\hat{p}$ is a empirical distribution, which is $\frac{1}{N}$ at each data point and zero else and p is the distribution generated by the CRBM. This expectation is intractable, because on has to go over $|\{0,1\}^{d+c}|$ states. But one can

still sample according to this distribution using alternate Gibbs sampling, i.e. iteratively sampling $p(x, y|h)$ and $p(h|x, y)$), which can be visualised in Fig. 98 by the arrow labelled W. This will converge to an unbiased sample. ◇

[Missing: slide 12 - 14]

General framework: Energy-based learning

So far we have seen two models for structured output: MDN and RBM. They are simple to train and the parameters of the RBM can be inspected to verify that the model has learned meaningful filters. But they also have limitation: the CRBM has a rigid product structure; there might be other functions that better model the input-output relation. The mixture density model captures the conditional distribution as a Gaussian distribution, but the true distribution might be better expressed by another distribution.

So far, we have only considered outputs that are real-valued vectors, while in practice we might want to predict more complicated structures such as trees or sequences.

Energy-based learning [LCH+06] is a general framework for performing structured predictions with neural networks. A energy function has two inputs: the observed variables and the variables Y to be predicted (very similar to the structured prediction kernel). For a given input there are different energy scores to possible predicted outputs (here: classes). The predicted output will be the class with the lowest energy, in this case: animal.

The model can be trained using a push/pull-approach.

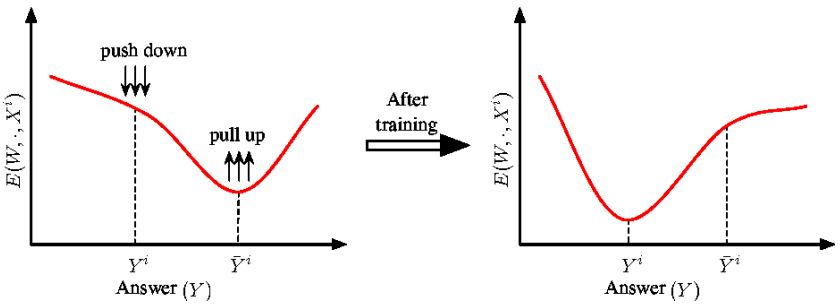


Fig. 101: For the correct answer Y^i, one wants the energy to be low and for the wrong one (so any other answer that is incorrect) to be high. We receive pairs of outputs, a correct and a incorrect one and one adapts the model W to increase/decrease the energy function accordingly. [LCH+06]

To achieve this training, one can use the generalised perceptron loss function

$$L(Y^i, E(W, Y, X^i)) := E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i.$$

But this loss function has a margin of zero: it halts as soon as one has produced a valid decision boundary, i.e. $E(W, Y^i, X^i) = \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$.

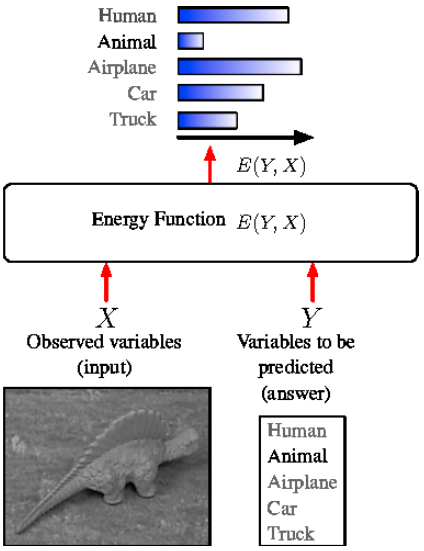


Fig. 99: A model measures the compatibility between observed variables X and variables to be predicted Y using an energy function $E(Y, X)$. For example, X could be the pixels of an image, and Y a discrete label describing the object in the image. Given X, the model produces the answer Y that minimises the energy E. [LCH+06]

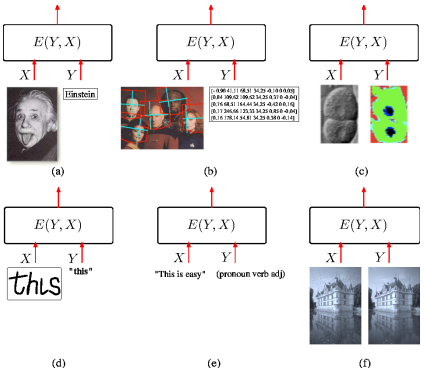


Fig. 100: Several applications of energy-based learning: (a) face recognition: Y is a high-cardinality discrete variable: an input image is detected to be Einstein or not Einstein; (b) face detection and pose estimation: Y is compromised of bounding boxes: a collection of vectors with location and pose of each possible face; (c) image segmentation: Y is an image in which each pixel is a discrete label (belonging to the nuclei or not); (d-e) handwriting recognition and sequence labelling: Y is a sequence of symbols from a highly structured but potentially infinite set (the set of English sentences). The situation is similar for many applications in natural language process-

The hinge loss forces this difference to be at least m.

Loss (equation #)	Formula	Margin
energy loss (6)	$E(W, Y^i, X^i)$	none
perceptron (7)	$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$	0
hinge (11)	$\max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	m
log (12)	$\log\left(1 + e^{E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)}\right)$	> 0

Fig. 102: Examples of loss functions. Margin is an important component to ensure that the learned model generalises well to test data.

Notice that the hinge loss looks like a ReLU and the log loss like a softplus. The log loss is desirable for complex energy functions.

How can we find the contrastive examples Y for which the loss function is high?

Missing: slides 19 - 21.

Example 4.2.4 (Structured Prediction: Kernels vs Networks)
One advantage of kernel-based structured prediction is that the margin is maximised in a well-defined feature space $\varphi(x)$ (margin corresponds to distance in EUCLIDEAN terms), whereas for neural networks, the margin is defined on some representation, which might distort the geometry of the input data, which is out of your control. Furthermore, the learning algorithm is convex, yielding reproducibility, whereas neural networks are nonconvex.

On the other hand, neural network based structured prediction grants more flexibility by letting the model extract the representation it needs for the task. Furthermore, the output space $\mathcal{Y}$ can be continuous (e.g. images) and inference of $y|x$ can be done via gradient descent (not directly in the output space but using e.g. a generator). ◇

In summary, like kernel machines, neural networks need to be adapted to be able to perform structure predictions. Simple methods for structured output learning include MDNs and CRBMs. More general structured predictions (e.g. sequences, trees) can be achieved within the flexible framework of energy-based learning.

4.3 Explainable Models

5 Federated Learning

References

[BC04] Tijl De Bie and Nello Cristianini, Kernel methods for exploratory pattern analysis: A demonstration on text data, SSPR/SPR, 2004.

[BPH+12] F. Bießmann, J. Papaioannou, A. Harth, M. Jugel, K. Müller, and M. Braun, Quantifying spatiotemporal dynamics of twitter replies to news feeds, 2012 IEEE International Workshop on Machine Learning for Signal Processing, 2012, pp. 1–6.

[BPT+20] Philipp Baumeister, Sebastiano Padovan, Nicola Tosi, Grégoire Montavon, Nadine Nettelmann, Jasmine MacKenzie, and Mareike Godolt, Machine-learning inference of the interior structure of low-mass exoplanets, The Astrophysical Journal **889** (2020), no. 1, 42.

[dai] `http://vlsi-design-engineers.blogspot.com/2015/10/exclusive-or-gates-parity-circuits-and.html`.

[EBK+20] Oliver Eberle, Jochen Büttner, Florian Kräutli, Klaus-Robert Müller, Matteo Valleriani, and Grégoire Montavon, Building and interpreting deep similarity models, arXiv preprint arXiv:2003.05431 (2020).

[Fuk80] Kunihiko Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics **36** (1980), 193–202.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl, Neural message passing for quantum chemistry, ArXiv **abs/1704.01212** (2017).

[HLMS04] Jihun Ham, Daniel Lee, Sebastian Mika, and Bernhard Schölkopf, A kernel view of the dimensionality reduction of manifolds, 07 2004.

[HOT36] HAROLD HOTELLING, RELATIONS BETWEEN TWO SETS OF VARIATES*, Biometrika **28** (1936), no. 3-4, 321–377.

[HZKM02] Stefan Harmeling, Andreas Ziehe, Motoaki Kawanabe, and Klaus-Robert Müller, Kernel feature spaces and nonlinear blind souce separation, Advances in neural information processing systems, 2002, pp. 761–768.

[JyHL+14] How Jing, Ting yao Hu, Hung-Shin Lee, Wei-Chen Chen, Chi-Chun Lee, Yu Tsao, and Hsin-Min Wang, Ensemble of machine learning algorithms for cognitive and physical speaker load detection, INTERSPEECH, 2014.

[KW16] Thomas N Kipf and Max Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[L+89] Yann LeCun et al., Generalization and network design strategies, Connectionism in perspective **19** (1989), 143–155.

[LCH+06] Yann LeCun, Sumit Chopra, Raia Hadsell, Aurelio Ranzato, and Fu Jie Huang, A tutorial on energy-based learning, 2006.

[LM04] Julian Laub and Klaus-Robert Müller, Feature discovery in non-metric pairwise data, J. Mach. Learn. Res. **5** (2004), 801–818.

[MH08] Laurens van der Maaten and Geoffrey Hinton, Visualizing data using t-sne, Journal of machine learning research **9** (2008), no. Nov, 2579–2605.

[MZKM02] Frank C. Meinecke, Andreas Ziehe, Motoaki Kawanabe, and Klaus-Robert Müller, A resampling approach to estimate the stability of one-dimensional or multidimensional independent components, IEEE Transactions on Biomedical Engineering **49** (2002), 1514–1525.

[NL] `http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg`.

[PCA] `https://www.astroml.org/book_figures/chapter7/fig_S_manifold_PCA.html`.

[RS00] Sam T Roweis and Lawrence K Saul, Nonlinear dimensionality reduction by locally linear embedding, science **290** (2000), no. 5500, 2323–2326.

[Sch06] Matthias Scholz, Approaches to analyse and interpret biological profile data, PhD thesis, University of Potsdam, 2006.

[SEL+20] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof Schütt, Klaus-Robert Müller, and Grégoire Montavon, Xai for graphs: Explaining graph neural network predictions by identifying relevant walks, ArXiv (2020).

[SPW+13] Richard Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, and C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, EMNLP **1631** (2013), 1631–1642.

[SSP+07] Sören Sonnenburg, Gabriele Schweikert, Petra Philips, Jonas Behr, and Gunnar Rätsch, Accurate splice site prediction using support vector machines, BMC bioinformatics, vol. 8, Springer, 2007, p. S7.

[SWSR08] Gabriele Schweikert, Christian Widmer, Bernhard Schölkopf, and Gunnar Rätsch, An empirical analysis of domain adaptation algorithms for genomic sequence analysis., vol. 8, 01 2008, pp. 1433–1440.

[VdMH12] Laurens Van der Maaten and Geoffrey Hinton, Visualizing non-metric similarities in multiple maps, Machine learning **87** (2012), no. 1, 33–55.

[WPC+20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long,

Chengqi Zhang, and S Yu Philip, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems (2020).

[WZM+00] G. Wubbeler, A. Ziehe, B. . Mackert, K. . Muller, L. Trahms, and C. Curio, Independent component analysis of noninvasively recorded cortical magnetic dc-fields in humans, IEEE Transactions on Biomedical Engineering **47** (2000), no. 5, 594–599.

[YNDT18] Rikiya Yamashita, Mizuho Nishio, Richard K. G. Do, and Kaori Togashi, Convolutional neural networks: an overview and application in radiology, Insights into Imaging **9** (2018), 611 – 629.

Index

C

convolution kernel 34

D

differential entropy 29

E

eigendecomposition 2
empirical cross-covariance matrices 17

I

IsoMap 9

L

linear kernel PCA 2

M

Metric MDS 9
model-induced kernels 39
Multiple Maps t-SNE 16
mutual information 29

P

parse tree 38

R

reconstruction error 50
residuals 50

S

Sammon mapping 9
Shannon-Entropy 28
stochastic neighbour selection 10

T

t-SNE 12
tokenisation approach 37

W

whitening transformation 26