Computational Intelligence Lab

Lecture Notes 2022

Thomas Hofmann (with updates by Gunnar Rätsch)
ETH Zurich

Copyright © 2021 Thomas Hofmann, 2022 Gunnar Rätsch
Unpublished manuscript for ETH-internal use

www.da.inf.ethz.ch/teaching/2022/CIL/

Contents

Preface

Everything should be made as simple as possible, but no simpler.
– Albert Einstein

Data science and machine learning have become the foundation for technologies and sciences that make use of data to solve a wide range of challenges such as interpreting experimental data, making predictions about likely outcomes, supporting data-informed decisions, and enabling machines to perform intelligent tasks, e.g. in machine vision and language understanding. There is today an evergrowing number of methods and models available, conveniently packaged in software libraries and powered by unprecedented computing machinery. It has never been easier to make use of methods of computational intelligence. However, this ease of use is not always accompanied by thorough understanding and a sufficient degree of conceptualization. The latter is problematic as the use of off the shelf tools without a sufficient level of competence may result in unreliable outcomes or lead to incorrect conclusions. Lack of understanding is clearly also a hindrance for the innovative use and advancement of methodology, be it in the core of machine intelligence, or in one of its many relevant domains of applications.

The goal of the *Computational Intelligence Lab* is to enable master level students to connect their mathematical background in linear algebra, analysis, probability, and optimization with their basic knowledge in machine learning and their general skill set in Computer Science to gain a deeper understanding of models and tools of great practical impact. This includes the often underestimated step of conceptualization and *critical modeling* of the problem at hand, i.e. reflecting on assumptions and simplifications and justifying the appropriateness of the approach taken. It also includes *replacing computation by calculation* where possible. It is very hard to understand what may happen, when we *run code over data* so to speak. What biases are introduced? What guarantees can be made? When will the method work, when fail? What would we even look for empirically to measure success? To answer such crucial questions, we need a *mathematical model* and not just a computational toolbox in which the model remains opaque to our understanding.

This follows the example set by physics, an area in which mathematical modeling plays a pivotal role, yet where the mathematical proficiency is not so much in the ability to prove fundamental theorems, but in doing back-of-the envelop calculations – sometimes on oversized envelops – that capture the essence of the problem, possibly in a sketchy manner. There may then remain a leap of faith towards reality, in our case, non-linear models with a degree of complexity that often escapes the rigor that we may ideally strive for. However, this does not devalue the need of applied mathematics in machine intelligence, quite the contrary.

These lecture notes are not an encyclopedia of mathematical modeling for machine intelligence. Rather, they are a compilation of relevant, weakly interconnected topics for which we carry out the program described above in an exemplary manner. The focus is not on breadth and there is no claim to any level of completeness, rather the aim is to go deep on a few selected topics. The topics chosen will (inevitably) overlap with other machine learning courses, yet our approach will differ. The goal here is not so much to teach new methods or models, but to train students in the competent use of mathematics, so that they can independently apply the learned skills to the models not covered. We will not provide a systematic introduction to the mathematics needed. One can consult excellent undergraduate textbooks or machine-inspired textbooks such as [8]. As far as programming goes, we will make use of `Python`, its scientific computing library `NumPy` and some more specialized libraries such as `PyTorch` when it comes to neural network models.

Syllabus 2022

02/25	Intro, Class Overview & Dimension Reduction I
03/04	Dimension Reduction II
03/11	Dimension Reduction III
03/18	Matrix Approximation I
03/25	Matrix Approximation II
04/01	Matrix Approximation III
04/08	Latent Variable Models I
04/29	Latent Variable Models I
05/06	Neural Networks I
05/13	Neural Networks II
05/20	Generative Models I
05/28	Generative Models II
06/03	PROJECT WORK

1. Dimension Reduction

1.1 Introduction

1.1.1 Motivation

The canonical representation for data is often in the form of vectors of measurements $\mathbf{x} \in \mathbb{R}^n$. The dimensionality can be low, if features are carefully chosen *a priori*, but is often very high in modern machine learning when raw data (e.g. images, audio, time-series) are used as input. It then becomes an intermediate goal of its own to find low dimensional representations, which compress the data, while preserving its relevant content. Even more, one often seeks representations that are interpretable and factor out different modes of variation. This theme has been prevalent in pattern recognition since the invention of principal component analysis by Pearson in 1901 [33].

In the absence of a task-specific notion of relevance, we will first study the objective of lossy reconstruction in a model class known as *autoencoders*. Figure 1.1 shows this in the graphical language of neural networks.

1.1.2 Setting

We consider the following setting:

Data Law. Data vectors are generated at random, following a probability distribution or data law ν. Typically ν is unknown and we only have access to a sample set $\mathcal{S} = \{\mathbf{x}_t \overset{\text{iid}}{\sim} \nu,\ t = 1, \dots, s\}$. We then write

$$\mathbf{E}_\nu[f(\mathbf{x})] = \int_{\mathbb{R}^n} f(\mathbf{x})\ \nu(d\mathbf{x}) \quad \text{or} \quad \mathbf{E}_\mathcal{S}[f(\mathbf{x})] = \frac{1}{s}\sum_{t=1}^{s} f(\mathbf{x}_t). \tag{1.1}$$

For instance, $\mathbf{E}_\mathcal{S}[\mathbf{x}]$ is the sample mean. We will drop the subscript, if the data law or the sample are unambiguous.

Encoder & Decoder An autoencoder consists of a complementary pair of maps: an encoder $F : \mathbb{R}^n \to \mathbb{R}^m$ and a decoder $G : \mathbb{R}^m \to \mathbb{R}^n$, which together form the *reconstruction map* $G \circ F : \mathbb{R}^n \to \mathbb{R}^n$. The ideal autoencoder would have $G \circ F = \mathrm{Id}$

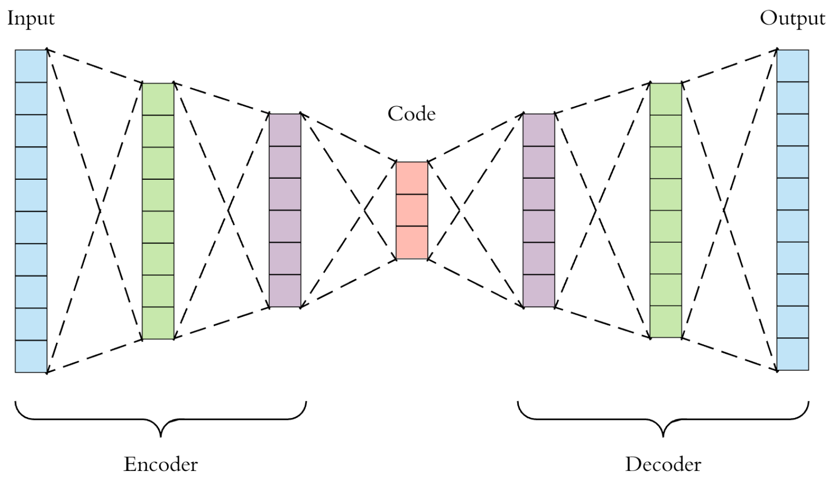


Figure 1.1: Autoencoder class of models for lossy reconstruction.

or $G = F^{-1}$, however this would not be interesting and the key idea is to restrict the dimensionality of the encoding by choosing $m < n$ (think: $m \ll n$). This forces the autoencoder to learn a compressed or dimension reduced data representation.

Reconstruction Loss As the autoencoder output may not perfectly reproduce the input, we need to specify a *distortion* or *loss function*, which is a function with signature $\ell : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. We will mainly focus on the quadratic loss $\ell(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{2}\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ between a pattern $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. The objective we aim to minimize is typically the expected loss (also called *risk*) of a reconstruction map $\mathcal{R}(H) = \mathbf{E}\left[\ell(\mathbf{x}, H(\mathbf{x}))\right]$, where $H = G \circ F$ in the autoencoder.

1.1.3 Challenge

In the spirit of this class, we would like to *understand* the autoencoder through *mathematical analysis*. We aim to generate insights about the model structure and biases, the role of the learning objective, the effectiveness of learning algorithms, and the dependencies on the data law. We will – to a large part – focus on the *linear* autoencoder, which we can analyse with the help of tools from linear algebra and multivariate calculus. The focus is less on *know how* (to implement it) as it is on *know what* (one does, if one implements or uses it). We will proceed in small steps towards what is known as Principal Component Analysis (PCA) and will conclude with a simple experiment on non-linear autoencoders.

As a side note:

[The method is a set of] reliable rules which are easy to apply, and such that if one follows them exactly, one will never take what is false to be true or fruitlessly expend one's mental efforts, but will gradually and constantly increase one's knowledge till one arrives at a true understanding of everything within one's capacity.

– René Descartes (Discours de la méthode)

1.2 Linear Autoencoder

The linear autoencoder is obtained by identifying F, G with linear maps

linear encoder	$F : \mathbf{x} \mapsto \mathbf{z} = \mathbf{W}\mathbf{x},$	$\mathbf{W} \in \mathbb{R}^{m \times n}\,.$
linear decoder	$G : \mathbf{z} \mapsto \hat{\mathbf{x}} = \mathbf{V}\mathbf{z},$	$\mathbf{V} \in \mathbb{R}^{n \times m}\,.$

which together with the squared loss leads to the risk

$$\mathcal{R}(\mathbf{W}, \mathbf{V}) = \mathcal{R}(\mathbf{P} := \mathbf{V}\mathbf{W}) = \mathbf{E}\left[\tfrac{1}{2}\|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2\right]\,. \tag{1.2}$$

Linear models are often a good starting point, both as a practical benchmark as well as an object of analysis, since *linearity* comes with a powerful arsenal of analytic tools. In the language of neural networks, $\mathbf{W}, \mathbf{V}$ are called weight matrices of the first (hidden) and second (output) layer, respectively.

1.2.1 Linear Compositionality

We start with a first observation: If we compose linear maps, the resulting map is linear. When representing linear maps by matrices, composition of maps is nothing but multiplication of matrices, the result of which is a new matrix.

→ The reconstruction map of the linear autoencoder is a linear map as linear maps are closed under composition. In matrix form it is given by $\mathbf{P} = \mathbf{V}\mathbf{W}$.

1.2.2 Linear vs. Affine Maps

One could consider a slightly more powerful class of models with affine (and not just linear) maps as encoder and/or decoder. However, this will potentially complicate implementation as well as analysis. It is important to understand this design choice.

Q Are affine maps more powerful than linear maps in autoencoders?

Definition 1.2.1 — Centering. The data is centered, if $\mathbf{E}\mathbf{x} = \mathbf{0}$. Data can be centered by transforming $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{E}\mathbf{x}$, i.e. by subtracting the mean.

Proposition 1.2.1 For centered data and the squared loss: optimal affine reconstruction maps are linear.

Proof. Let $\mathbf{a} \neq \mathbf{0}$, then
$\mathbf{E}\|\mathbf{x} - (\mathbf{P}\mathbf{x} + \mathbf{a})\|^2 = \mathbf{E}\|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2 + \|\mathbf{a}\|^2 - 2\langle \mathbf{a}, \underbrace{\mathbf{E}\mathbf{x} - \mathbf{P}\mathbf{E}\mathbf{x}}_{=\mathbf{0}}\rangle > \mathbf{E}\|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2.$ ■

Corollary 1.2.2 For centered data, optimal affine maps degenerate to linear ones.

Proof. Note that $\mathbf{V}(\mathbf{W}\mathbf{x} + \mathbf{a}) + \mathbf{b} = \mathbf{V}\mathbf{W}\mathbf{x} + \mathbf{c}$, where $\mathbf{c} = \mathbf{b} + \mathbf{V}\mathbf{a}$. ■

→ When centering data as a preprocessing step, affine maps cannot obtain better reconstruction than linear ones in autoencoders.

1.2.3 Non-Identifiability

As the expected loss only depends on the product of the weight matrices $\mathbf{P} = \mathbf{VW} \in \mathbb{R}^{n \times n}$, one can ask:

Q Is the representation of $\mathbf{P}$ as $\mathbf{P} = \mathbf{VW}$ unique?

This is also called (parameter) identifiability. Note that there are two levels of uniqueness: First, is the optimal linear reconstruction map $\mathbb{R}^n \to \mathbb{R}^n$ unique? Second, is the parameterization via weight matrices $\mathbf{W}, \mathbf{V}$ unique? In the linear autoencoder, obviously $\mathbf{VW} = \mathbf{VIW}$, so we can 'squeeze in' any pair of matrices between $\mathbf{V}$ and $\mathbf{W}$ that yield the identity $\mathbf{I}$, more precisely:

Proposition 1.2.3 For any solution pair $(\mathbf{W}, \mathbf{V})$

$$\{(\mathbf{W}', \mathbf{V}') : \mathbf{V}'\mathbf{W}' = \mathbf{VW}\} \supseteq \{(\mathbf{W}', \mathbf{V}') : \mathbf{W}' = \mathbf{AW},\ \mathbf{V}' = \mathbf{VA}^{-1},\ \mathbf{A} \text{ invertible}\}.$$

Proof. $\mathbf{V}'\mathbf{W}' = \mathbf{VA}^{-1}\mathbf{AW} = \mathbf{VW}$. ■

Corollary 1.2.4 $\mathcal{R}(\mathbf{W}, \mathbf{V}) = \mathcal{R}(\mathbf{AW}, \mathbf{VA}^{-1})$ for any invertible $\mathbf{A}$.

→ The weight matrices are non-identifiable and one needs to be careful not to over-interpret the found representation.

As a consequence of the non-identifiability, we may want to investigate constrained classes of square matrices $\mathbf{P}$ and postpone the question of how to split $\mathbf{P} = \mathbf{VW}$ (non-uniquely) into a product of weight matrices. But this means we need to first understand the constraint imposed by the bottleneck architecture on the reconstruction map.

1.2.4 Rank-Constraint

Q What is the structural constraint on the reconstruction map in the autoencoder, i.e. how can we characterize $\mathbf{P} = \mathbf{VW}$?

Recall that $\mathbf{V} \in \mathbb{R}^{n \times m}$ and $\mathbf{W} \in \mathbb{R}^{m \times n}$. We want m to be small relative to n, possibly $m \ll n$. It is immediately clear[1] that $\dim(\text{im}(\mathbf{W})) \leq m$ as:

Proposition 1.2.5 For a linear map L represented by $\mathbf{A}$, $\text{im}(L) = \text{span}\{\text{columns of } \mathbf{A}\}$.

Proof. Follows from the definition of the matrix-vector product. ■

Definition 1.2.2 — Rank. The rank of a linear map L is $\text{rank}(L) = \dim(\text{im}(L))$.

Proposition 1.2.6 $L : V \to V'$ linear, $\dim(V) = n$, $\dim(V') = m$: $\text{rank}(L) \leq \min\{m, n\}$.

Proof. Obvious, if $m \leq n$ as $\text{im}(L) \subseteq V'$. If $n < m$, by linearity and definition of image we have $\text{im}(L) = \text{span}\{L(\mathbf{v}_1), \ldots, L(\mathbf{v}_n)\}$ for any choice of basis of V. The dimensionality of the span of n vectors cannot exceed n. ■

[1] By $\text{im}(\mathbf{W})$ we mean the image of the linear map with matrix representation $\mathbf{W}$ in the canonical basis.

Proposition 1.2.7 Given linear maps $L : V \to V'$, $L' : V' \to V''$, then

$$\operatorname{rank}(L' \circ L) \leq \min\{\operatorname{rank}(L), \operatorname{rank}(L')\}.$$

Proof. Thanks to Proposition 1.2.6, we just need to show that $\operatorname{rank}(L' \circ L) \leq \dim(V')$. Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ be a basis for V'; by linearity of L', $\operatorname{im}(L')$ is $\operatorname{span}\{L'(\mathbf{v}_1), \dots, L'(\mathbf{v}_m)\}$, which has at most dimension m. Since $\operatorname{im}(L' \circ L) \subseteq \operatorname{im}(L')$, $\operatorname{rank}(L' \circ L) \leq m$. ■

We conclude with the following corollaries.

Corollary 1.2.8 If $\operatorname{im}(L) \cap \ker(L') = \{\mathbf{0}\}$, then equality holds in Proposition 1.2.7.

Corollary 1.2.9 For matched matrices $\mathbf{A}$, $\mathbf{B}$: $\operatorname{rank}(\mathbf{AB}) = \min\{\operatorname{rank}(\mathbf{A}), \operatorname{rank}(\mathbf{B})\}$.

→ The reconstruction map of a linear autoencoder is of rank less or equal to m. The bottleneck layer constitutes a rank constraint.

1.3 Projections

Note that the rank limitation on $\mathbf{P}$ is independent of the loss function and so is the lack of identifiability of its factorization. We will now exploit the specifics of the squared loss. The rank constraint together with linearity implies that $\operatorname{im}(\mathbf{P})$ is a linear subspace $U \subseteq \mathbb{R}^n$ of dimension at most m.

Q Given a subspace $U \subseteq \mathbb{R}^n$, what is the optimal linear reconstruction map $\mathbf{P}$ such that $\mathcal{R}(\mathbf{P})$ is minimized subject to $\operatorname{im}(\mathbf{P}) = U$?

1.3.1 Orthogonal Projection

We can define the optimal reconstruction map implicitly.

Definition 1.3.1 The orthogonal projection to a linear subspace $U \subseteq \mathbb{R}^n$ is defined as

$$\Pi_U : \mathbb{R}^n \to U, \quad \Pi_U(\mathbf{x}) = \arg\min_{\mathbf{x}' \in U} \|\mathbf{x} - \mathbf{x}'\|^2.$$

As standard, we call $U^\perp$ the linear subspace $\{\mathbf{x}^\perp \in \mathbb{R}^n \mid \langle \mathbf{x}^\perp, \mathbf{x}' \rangle = 0, \forall \mathbf{x}' \in U\}$.

Proposition 1.3.1 For each $\mathbf{x} \in \mathbb{R}^n$ and each linear subspace U, $\Pi_U(\mathbf{x})$ exists and is unique. Moreover, $\Pi_U(\mathbf{x})$ is the only vector in U such that $\Pi_U(\mathbf{x}) - \mathbf{x} \in U^\perp$ (i.e. Π_U is orthogonal).

Proof. The optimization problem in the definition of projection always has a solution, since $\mathbf{0} \in U$. For each $\mathbf{x}$, the solution is unique because there exists a unique minimum of the objective. Let $\Pi_U(\mathbf{x}) - \mathbf{x} = \hat{\mathbf{x}} + \hat{\mathbf{x}}^\perp$ with $\hat{\mathbf{x}} \in U$, $\hat{\mathbf{x}}^\perp \in U^\perp$. By the Pythagorean theorem

$$\|\Pi_U(\mathbf{x}) - \mathbf{x}\|^2 = \|\hat{\mathbf{x}}\|^2 + \|\hat{\mathbf{x}}^\perp\|^2 \geq \|\hat{\mathbf{x}}^\perp\|^2 = \|\underbrace{\Pi_U(\mathbf{x}) - \hat{\mathbf{x}}}_{\in U} - \mathbf{x}\|^2.$$

which by the definition of Π_U implies $\hat{\mathbf{x}} = \mathbf{0}$ and thus the claim. ■

Proposition 1.3.2 Π_U is linear.

Proof. We need to show homogeneity and additivity of the map Π_U:

$$\begin{aligned}\Pi_U(\alpha\mathbf{x}) &= \arg\min_{\alpha\mathbf{x}'} \|\alpha\mathbf{x}-\alpha\mathbf{x}'\| = \alpha \arg\min_{\mathbf{x}'} \|\mathbf{x}-\mathbf{x}'\| = \alpha\Pi_U(\mathbf{x})\\ \Pi_U(\mathbf{x}+\mathbf{y}) &= \Pi_U(\mathbf{x}) + \arg\min_{\mathbf{y}'\in U} \|\Pi_U(\mathbf{x})+\mathbf{y}'-(\mathbf{x}+\mathbf{y})\|^2\\ &= \Pi_U(\mathbf{x}) + \arg\min_{\mathbf{y}'\in U}\{\|\mathbf{y}-\mathbf{y}'\|^2 + 2\underbrace{\langle \Pi_U(\mathbf{x})-\mathbf{x},\mathbf{y}'\rangle}_{=\langle U^\perp,U\rangle=0}\} = \Pi_U(\mathbf{x})+\Pi_U(\mathbf{y}).\end{aligned}$$

The first equality uses the identity $\arg\min_{\mathbf{y}'} f(\mathbf{y}') = \mathbf{y}_0 + \arg\min_{\mathbf{y}'} f(\mathbf{y}_0+\mathbf{y}')$ for $\mathbf{y}_0 = \Pi_U(\mathbf{x})$. In the second equality we have multiplied out the square and dropped terms that are independent of $\mathbf{y}'$ (and hence do not change the arg min). ■

→ For given subspace U the optimal reconstruction map $\mathbf{P}$ is the matrix representing the orthogonal projection Π_U. The optimal linear autoencoder represents a projection.

1.3.2 Orthogonal Projection Matrices

Q How can we explicitly represent projections as matrices and what representation can we extract from a solution of the autoencoder?

Example 1.1 The orthogonal projection to the line corresponding to the unit vector $\mathbf{u}$ can be represented as the matrix

$$\mathbf{P}_\mathbf{u} = \mathbf{u}\mathbf{u}^\top, \quad \|\mathbf{u}\|^2 = \langle\mathbf{u},\mathbf{u}\rangle = 1.$$

Note that by associativity $(\mathbf{u}\mathbf{u}^\top)\mathbf{x} = \langle\mathbf{u},\mathbf{x}\rangle\,\mathbf{u}$. The inner product represents the signed length of the projected vector,

$$|\langle\mathbf{u},\mathbf{x}\rangle| = |\cos\angle(\mathbf{u},\mathbf{x})|\,\|\mathbf{x}\| \le \|\mathbf{x}\|\,.$$

■

We can generalize this example to linear subspaces of arbitrary dimension.

Proposition 1.3.3 For any linear subspace $U \subseteq \mathbb{R}^n$ pick an orthonormal basis $\{\mathbf{u}_1,\dots\mathbf{u}_k\}$ and define

$$\mathbf{P} = \mathbf{U}\mathbf{U}^\top, \quad \mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_k\end{bmatrix}, \quad \mathbf{P}\mathbf{x} = \sum_{i=1}^k \langle\mathbf{u}_i,\mathbf{x}\rangle\mathbf{u}_i\,.$$

Then $\mathbf{P}$ is a matrix representation of the orthogonal projection Π_U.

Proof. Clearly $\mathrm{im}(\mathbf{P}) = U$ as the columns of $\mathbf{P}$ span U. One can verify orthogonality via

$$\langle\mathbf{P}\mathbf{x},\mathbf{x}-\mathbf{P}\mathbf{x}\rangle \overset{*1}{=} \langle\mathbf{x},\mathbf{P}\mathbf{x}-\mathbf{P}^2\mathbf{x}\rangle = \langle\mathbf{x},(\mathbf{P}-\mathbf{P}^2)\,\mathbf{x}\rangle \overset{*2}{=} 0, \quad (\forall\mathbf{x}).$$

The equalities $*_1$ and $*_2$ can be identified as self-adjointness (symmetry) and idempotency properties, respectively. Their proof is left as an exercise (see below). ■

→ The proposition gives an explicit representation of the projection matrix for a subspace U via choice of an orthonormal basis.

Definition 1.3.2 — Projection, Idempotence. A linear map $P : V \to V$ is a projection, if it is idempotent, i.e. $P^2 = P$.

R If P is represented as a matrix $\mathbf{P}$, then $\mathbf{P}^2 = \mathbf{P} \cdot \mathbf{P} = \mathbf{P}$.

Definition 1.3.3 — Self-Adjoint. A linear map $P : V \to V$ over an inner product space is self-adjoint if $\langle P(\mathbf{x}), \mathbf{y} \rangle = \langle \mathbf{x}, P(\mathbf{y}) \rangle$ $(\forall \mathbf{x}, \mathbf{y} \in V)$.

R If P is represented as a matrix $\mathbf{P}$ over $\mathbb{R}$ (or $\mathbb{C}$), then $\mathbf{P}$ is symmetric (or Hermitian).

The concepts of idempotency and self-adjointness also characterize orthogonal projections in the infinite dimensional case.

Proposition 1.3.4 — Hilbert Projection Theorem. A projection $P : V \to V$ over a Hilbert space (complete inner product space) is orthogonal, if it is self-adjoint.

Exercise 1.1 The orthogonal projection matrix $\mathbf{P}$ is self-adjoint $(*_1)$ and idempotent $(*_2)$ (cf. Proposition 1.3.3). ■

While the optimal $\mathbf{P} = \mathbf{V}\mathbf{W}$ is a projection matrix, it may not be represented in the orthonormal form of Proposition 1.3.3. However, if we enforce parameter sharing via $\mathbf{V} = \mathbf{W}^\top$, then the optimal weight matrix will have to be an orthonormal basis.

Exercise 1.2 If $\mathbf{P}$ is a projection matrix and $\mathbf{P} = \mathbf{W}^\top \mathbf{W}$ then $\mathbf{W}$ is orthogonal. ■

→ The optimal weight matrix of the autoencoder with tied parameter matrices $\mathbf{V} = \mathbf{W}^\top$ is orthogonal.

1.3.3 Oblique Projection Matrices

Without parameter tying, an optimal solution of the autoencoder will still represent a projection matrix, yet it will not necessarily factorize as $\mathbf{P} = \mathbf{U}\mathbf{U}^\top$, with $\mathbf{U}$ orthogonal.

Q What will the optimal weight matrices of the autoencoder represent without parameter tying?

In a non-orthonormal basis $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ for U one can recover the projection matrix:

Proposition 1.3.5 Define $\mathbf{V} = [\mathbf{v}_1 \dots \mathbf{v}_m]$ for linearly independent vectors, then

$$\mathbf{P} = \mathbf{V}\mathbf{V}^+, \quad \mathbf{V}^+ := \left(\mathbf{V}^\top \mathbf{V}\right)^{-1} \mathbf{V}^\top,$$

is the projection matrix for the subspace $U = \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$.

Proof. Let $\mathbf{P}$ be the projection matrix of U. Then $\mathbf{P}\mathbf{v}_i = \mathbf{v}_i$ and thus $\mathbf{PV} = \mathbf{V}$. As $\mathbf{P}$ has to be of rank $\le m$, the identity matrix is not an option. Define $\mathbf{P} = \mathbf{V}\mathbf{V}^+$, where $\mathbf{V}^+$ is the left Moore-Penrose pseudo-inverse of $\mathbf{V}$. Then

$$\mathbf{PV} = \mathbf{V}\mathbf{V}^+\mathbf{V} = \mathbf{V}[(\mathbf{V}^\top\mathbf{V})^{-1}\mathbf{V}^\top]\mathbf{V} = \mathbf{V}, \quad \mathbf{V}^+ \in \mathbb{R}^{m\times n} \quad (\text{i.e., rank}(\mathbf{V}^+) \le m)$$

The rank constraint implies $\mathbf{P}\mathbf{u}^\perp = 0$ for $\mathbf{u}^\perp \in U^\perp$, which yields the claim. ■

Corollary 1.3.6 For any $\mathbf{V}$: $\mathbf{V}^+ = \arg\min_{\mathbf{W}} \mathcal{R}(\mathbf{W}, \mathbf{V})$.

→ Given $\mathbf{V}$ the optimal choice of $\mathbf{W}$ will be the left pseudo-inverse of $\mathbf{V}$.

R Note that the columns of $\mathbf{V}$ span the subspace U to which the reconstructed data are confined. The pseudoinverse takes obliqueness of the columns into account and projects orthogonally to U.

Example 1.2 At this point, we want to include a code example in Python. Let us assume that we have some way of choosing $\mathbf{V}$. For example, we can consider it to be a matrix with iid random entries or we can define it by selecting m random data points.

```
# choice #1: random entries
V = np.random.rand(n,m)

# choice #2: random patterns (here: first m columns of data matrix)
V = X[:,:m]

# optimal choice for W
W = np.linalg.pinv(V)

# example: sklearn.datasets digits
>> Pseudo inverse, random projection, MSE = 0.6805339818228812
>> Pseudo inverse, random data columns, MSE = 0.39583375487949624
```

As we can see the 'statistical' randomization, which actually makes use of the data to chose $\mathbf{V}$ is superior. ■

1.4 Principal Component Analysis

1.4.1 Variance Maximization

We know that the optimal solution of the autoencoder will implement a projection to a subspace of dimension m or less, possibly parameterized in oblique form. Yet we are left with the key question:

Q Which subspaces U are optimal to project onto?

Note that so far, with the exception of centering, we have not made any reference to the data law ν. We took this route on purpose as it allows us to understand what the structure of the model and the choice of objective already implies *a priori* about its solutions. The remaining question is to analyze and possibly solve

$$\overset{\min}{\rightarrow} \mathbf{E}\|\mathbf{x}-\mathbf{P}\mathbf{x}\|^2, \quad \mathbf{P}=\mathbf{U}\mathbf{U}^\top \text{ projection matrix.}$$

Note that

$$\mathbf{E}\|\mathbf{x}-\mathbf{P}\mathbf{x}\|^2 - \underbrace{\mathbf{E}\|\mathbf{x}\|^2}_{const} = \mathbf{E}\langle\mathbf{P}\mathbf{x},\mathbf{P}\mathbf{x}\rangle - 2\mathbf{E}\langle\mathbf{x},\mathbf{P}\mathbf{x}\rangle = \mathbf{E}\langle\mathbf{x},(\mathbf{P}^2-2\mathbf{P})\mathbf{x}\rangle = -\mathbf{E}\langle\mathbf{x},\mathbf{P}\mathbf{x}\rangle,$$

where we have used self adjointness and idempotency. In summary:

Proposition 1.4.1 For centered data the reconstruction loss of a projection matrix $\mathbf{P}$ is $\frac{1}{2}(\mathrm{Var}[\mathbf{x}] - \mathrm{Var}[\mathbf{P}\mathbf{x}])$.

Proof. As above and noting[2] that $\mathbf{E}\|\mathbf{x}\|^2 = \mathrm{Var}[\mathbf{x}]$, $\mathbf{E}\langle\mathbf{x},\mathbf{P}\mathbf{x}\rangle = \mathbf{E}\langle\mathbf{P}\mathbf{x},\mathbf{P}\mathbf{x}\rangle = \mathrm{Var}[\mathbf{P}\mathbf{x}]$. ■

Corollary 1.4.2 For centered data, the optimal autoencoder represents the projection $\mathbf{P}$ which maximizes the variance.

→ The optimal autoencoder projects data so as to preserve as much as possible of their variance.

The above characterization makes use of an expectation over projected data, which (formally) corresponds to the use of the pushforward measure induced by $\mathbf{P}$. It is useful to express this in terms of statistics of the original data.

Q What is a sufficient statistics of the data?

Proposition 1.4.3 $\mathrm{Var}(\mathbf{P}\mathbf{x}) = \mathrm{tr}(\mathbf{P}\mathbf{E}[\mathbf{x}\mathbf{x}^\top])$.

Proof. It is elegant to make use of the trace operator, which is linear (commutes with expectation) and invariant under cyclic permutations of matrix products, specifically

$$\mathrm{Var}(\mathbf{P}\mathbf{x}) = \mathbf{E}\langle\mathbf{x},\mathbf{P}\mathbf{x}\rangle = \mathbf{E}\,\mathrm{tr}(\mathbf{x}^\top\mathbf{P}\mathbf{x}) = \mathbf{E}\,\mathrm{tr}(\mathbf{P}\mathbf{x}\mathbf{x}^\top) = \mathrm{tr}(\mathbf{P}\mathbf{E}[\mathbf{x}\mathbf{x}^\top]).$$

■

→ The optimal projection is fully determined by the covariance matrix of the data, i.e. $\mathbf{E}[\mathbf{x}\mathbf{x}^\top]$ are sufficient statistics (together with $\mathbf{E}[\mathbf{x}]$ used in centering).

[2]Here, we denote by $\mathrm{Var}[\mathbf{x}]$ the trace of the covariance matrix of $\mathbf{x}$.

1.4.2 Principal Components

Q How does the *optimal* projection relate to the data covariance matrix?

Let us spoil the suspense and state the answer.

Theorem 1.4.4 The rank m projection optimizing the squared reconstruction loss on centered data is given by the projection matrix formed by m orthogonal, principal eigenvectors of $\mathbf{E}[\mathbf{x}\mathbf{x}^\top]$.

Let us work towards a proof, starting from the diagonal case. Note that the covariance matrix is guaranteed to be positive definite and symmetric by construction.

Proposition 1.4.5 Let $\mathbf{E}[\mathbf{x}\mathbf{x}^\top] = \mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$, $\lambda_1 \geq \dots \geq \lambda_n \geq 0$. A set of m orthogonal, principal eigenvectors is given by the unit vectors $\{\mathbf{e}_1, \dots, \mathbf{e}_m\}$.

Proof. The unit vectors are orthogonal. They are eigenvectors of any diagonal matrix, specifically $\mathbf{\Lambda}\mathbf{e}_j = \lambda_j \mathbf{e}_j$. Clearly, the chosen ordering of the diagonal elements induces the claimed precedence on unit vectors. Note that in case of multiplicities, the choice of unit vectors may not be unique. ■

Proposition 1.4.6 As in Proposition 1.4.5. The following projection is variance maximizing

$$\mathbf{P} = \sum_{i=1}^{m} \mathbf{e}_i \mathbf{e}_i^\top = \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

Proof. We are seeking the maximizer of the variance

$$\max_{\mathbf{U}} \to \operatorname{tr}(\mathbf{U}\mathbf{U}^\top \mathbf{\Lambda}) = \sum_{i=1}^{n} \lambda_i \sum_{j=1}^{m} u_{ij}^2, \quad \text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbf{I}_m$$

1. We want to show that $\sum_{j=1}^m u_{ij}^2 \leq 1$ $(\forall i)$. Assume that it would not hold for i, then

   $$\|\mathbf{P}\mathbf{e}_i\|^2 = \mathbf{e}_i^\top \mathbf{P}^\top \mathbf{P}\mathbf{e}_i = \mathbf{e}_i^\top \mathbf{U}\mathbf{U}^\top \mathbf{e}_i = \sum_j u_{ij}^2 \overset{?}{>} 1,$$

   which contradicts the non-expansiveness of a projection (i.e. eigenvalues are in $\{0, 1\}$).
2. The Frobenius norm is given by $\sum_{ij} u_{ij}^2 = \operatorname{tr}(\mathbf{I}_m) = m$.
3. As the diagonal elements are ordered in decreasing magnitude any $\mathbf{U}$ such that $\sum_{j=1}^m u_{ij}^2 = 1$ for $i = 1, \dots, m$ would be optimal (if it exists).
4. The claimed form of $\mathbf{P}$ has exactly this extremal property.

■

Theorem 1.4.7 — Spectral Theorem. Any symmetric and positive semidefinite matrix $\mathbf{\Sigma}$ can be non-negatively diagonalized with an orthogonal matrix

$$\mathbf{\Sigma} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top, \quad \mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n), \quad \lambda_1 \geq \dots \geq \lambda_n \geq 0, \quad \mathbf{Q} \text{ orthogonal.}$$

Corollary 1.4.8 The variance maximizing projection for a covariance matrix $\mathbf{\Sigma}$ as in Theorem 1.4.7 is given by

$$\mathbf{P} = \mathbf{U}\mathbf{U}^\top, \quad \mathbf{U} = \mathbf{Q}\begin{pmatrix}\mathbf{I}_m \\ \mathbf{0}\end{pmatrix}.$$

Proof. By Theorem 1.4.7 and Proposition 1.4.5 and using cyclicity of the trace

$$\mathrm{tr}(\mathbf{U}\mathbf{U}^\top\mathbf{\Sigma}) = \mathrm{tr}((\mathbf{Q}^\top\mathbf{U})^\top\mathbf{\Lambda}(\mathbf{Q}^\top\mathbf{U})), \quad \text{maximized by} \quad \mathbf{Q}^\top\mathbf{U} = \begin{pmatrix}\mathbf{I}_m \\ \mathbf{0}\end{pmatrix}.$$

■

This concludes the proof of Theorem 1.4.4 by identifying the maximizer in Corollary 1.4.8 with a matrix build out of m principal vectors of $\mathbf{\Sigma} = \mathbf{E}[\mathbf{x}\mathbf{x}^\top]$. ■

→ The linear autoencoder network performs principal component projection. However, it is not guaranteed to identify the PCA basis (i.e. the principal eigenvectors), but only the m-dimensional principal subspace of $\mathbf{E}[\mathbf{x}\mathbf{x}^\top]$.

1.4.3 Eigenvectors of Symmetric Matrices

Q Why eigenvectors?

The eigenvectors played a prominent role in the analysis of this section, but one may wonder what is special about them? After all, they are defined in a way that seems unrelated to variance maximization. We focus here on the important subclass of symmetric matrices to confine the material.

As we have seen, symmetric matrices are representations of self-adjoint linear maps. Their diagonalization as expressed in the spectral theorem 1.4.7 essentially states that they are diagonal, when represented in the right choice of orthonormal basis. This is because orthogonal matrices are basis transformations between orthonormal bases.

Definition 1.4.1 — Eigenvector, Self-Adjoint. $\mathbf{u} \neq \mathbf{0}$ is an eigenvector of a symmetric matrix $\mathbf{A}$, if there exists a $\lambda \in \mathbb{R}$ such that $\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$.

The principal eigenvector is related to the 2-norm or spectral norm of a matrix.

Proposition 1.4.9 Let $\mathbf{A}$ be symmetric with principal (unit) eigenvector $\mathbf{u}$, i.e. $\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$, $\lambda = \max\{\lambda_{\max}(\mathbf{A}), -\lambda_{\min}(\mathbf{A})\}$, then

$$\|\mathbf{A}\mathbf{u}\| = \max_{\mathbf{v}:\|\mathbf{v}\|=1} \|\mathbf{A}\mathbf{v}\|.$$

Proof. This is easily checked in the diagonal case and remains true under change of orthonormal basis. ■

Note that if $\mathbf{\Sigma}$ is a covariance matrix, $\langle \mathbf{u}, \mathbf{\Sigma}\mathbf{u}\rangle$ is the variance of the projected real-valued random variable. Obviously, projecting on a principal eigenvector maximizes the variance.

The second important fact about eigenvectors of symmetric matrices is that they are essentially pairwise orthogonal.

Proposition 1.4.10 Eigenvectors of symmetric matrices with distinct eigenvalues are orthogonal.

Proof. Let $\mathbf{Au} = \lambda \mathbf{u}$, $\mathbf{Au}' = \lambda' \mathbf{u}'$, $\lambda \neq \lambda'$. Then

$$\lambda \langle \mathbf{u}, \mathbf{u}' \rangle = \langle \lambda \mathbf{u}, \mathbf{u}' \rangle = \langle \mathbf{Au}, \mathbf{u}' \rangle = \langle \mathbf{u}, \mathbf{Au}' \rangle = \langle \mathbf{u}, \lambda' \mathbf{u}' \rangle = \lambda' \langle \mathbf{u}, \mathbf{u}' \rangle \overset{\lambda \neq \lambda'}{\Longrightarrow} \langle \mathbf{u}, \mathbf{u}' \rangle = 0.$$

■

Example 1.3 As a counterexample: all vectors are eigenvectors of $\mathbf{I}$ with eigenvalue 1. ■

We can identify the second, third etc. subprincipal eigenvectors as principal eigenvectors of a sequence of matrices, which we obtain by projecting out the chosen direction.

$$\mathbf{A}' = \mathbf{A} - \lambda \mathbf{u}\mathbf{u}^\top \quad \text{s.t.} \quad \mathbf{A}'(\alpha \mathbf{u}) = \alpha \lambda \mathbf{u} - \alpha \lambda \mathbf{u} \|\mathbf{u}\| = \mathbf{0}. \tag{1.3}$$

In the (sorted) diagonal view, this simply zeros out the maximal eigenvalue.

$$\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \text{diag}(\gamma_1, \gamma_2, \ldots, \gamma_n) \mapsto \text{diag}(0, \gamma_2, \ldots, \gamma_n) = \mathbf{Q}^\top \mathbf{A}' \mathbf{Q}. \tag{1.4}$$

This is to say that the $(k+1)$-th subprincipal unit eigenvector maximizes the variance of the projection (or the norm of the image) subject to orthogonality to the first k principal eigenvectors. This procedural view is also true in the case of (geometric) multiplicities of eigenvalues. One can then simply chose any orthonormal basis for the respective subspaces spanned by eigenvectors with the same eigenvalue.

→ Unit eigenvectors of a symmetric, positive semidefinite matrix form an orthonormal basis in which the order of principality agrees with the magnitude of the norm of their image and – in the case of covariance matrices – the order of the variance of their projection.

1.5 Learning Algorithms

1.5.1 Matrix Computations

We can compute what the autoencoder aims to learn without having to make reference to the diagrammatic representation. There is a rich body of algorithms in the area of matrix computation [16] that can be used to diagonalize a symmetric positive semi-definite matrix. As this is specialized material from numerical linear algebra, we will not pursue this direction further. One can find implementations of methods in LAPACK, which can be used through the `numpy.linalg` library (e.g. `eigh`).

Example 1.4 There is a special function in numpy for computing eigenvectors of symmetric (or Hermitian) matrices. We have to select the last m columns.

```
covar            = np.cov(X)
evals, evecs     = np.linalg.eigh(covar)
V                = evecs[:,-m:]
W                = np.transpose(V)

>> Principal eigenvectors, MSE = 0.2658367979102716
```

■

1.5.2 Power Method

There is a very simple, yet scalable and easy to analyse algorithm for computing a principal eigenvector. Initializing at random $\mathbf{v}^{(0)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, one computes the iterate sequence

$$\mathbf{v}^{(t+1)} = \frac{\mathbf{A}\mathbf{v}^{(t)}}{\|\mathbf{A}\mathbf{v}^{(t)}\|}. \tag{1.5}$$

Proposition 1.5.1 Let $\mathbf{u}_1$ be the unique principal vector of a diagonalizable matrix $\mathbf{A}$ with eigenvalues $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Then the power iterates $\lim_{t\to\infty} \mathbf{v}^{(t)} = \mathbf{u}_1$ if $\langle \mathbf{v}_0, \mathbf{u} \rangle \neq 0$.

Proof. Represent $\mathbf{v}^{(0)}$ in the eigenbasis as $\mathbf{v}^{(0)} = \sum_i \alpha_i \mathbf{u}_i$, where $\alpha_1 \neq 0$. Then

$$\mathbf{v}^{(k)} \propto \alpha_1 \mathbf{u}_1 + \sum_{i>1} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{u}_i, \quad \text{where} \quad \frac{\lambda_i}{\lambda_1} < 1 \quad \text{for } i > 1\,.$$

Hence the sum converges to 0 and by virtue of the normalization $\mathbf{v}^{(k)} \to \mathbf{u}_1$. ■

Exercise 1.3 How can the power method be used (naïvely) to compute a principal subspace? ■

→ The power method is very simple and intuitive, yet often inefficient algorithm to identify the principal eigenspace of a symmetric matrix.

If the goal is to identify a principal eigenspace, one can improve on this by performing Arnoldi iterations, which can be further optimized for symmetric positive semidefinite matrices in Lanczos algorithm. The idea is to make use of intermediate results and to approximately and iteratively construct an eigenbasis. The idea is to form the Krylov matrix

$$\mathcal{K} = \begin{bmatrix} \mathbf{v} & \mathbf{A}\mathbf{v} & \mathbf{A}^2\mathbf{v} & \dots & \mathbf{A}^k\mathbf{v} \end{bmatrix}, \tag{1.6}$$

and to orthogonalize its vectors via a variant of the Gram-Schmidt process

$$\mathbf{v}_j = \mathbf{v}_j - \langle \mathbf{v}_j, \mathbf{v}_{k+1} \rangle, \quad j = 1, \dots, k \tag{1.7}$$

to get an orthogonal basis of the Krylov spaces. In each iteration one also keeps track of the inner products, which form a so-called upper Hessenberg matrix.[3]

Further Reading. The interested reader can find the precise convergence rates for the Lanczos algorithm and the Power Method in the seminal paper by Kuczyński and Woźniakowski: "*Estimating the largest eigenvalues by the power and Lanczos algorithms with a random start*" [29].

[3] `https://en.wikipedia.org/wiki/Arnoldi_iteration`

1.5.3 Gradient Descent

Another very generic and scalable approach is to treat the autoencoder as a neural network and use the workhorse of Deep Learning, namely stochastic gradient descent. At the core of these methods, one needs to compute the parameter gradient of the squared loss for a single data point.

$$\ell(\mathbf{x};\mathbf{P}) = \tfrac{1}{2}\langle \mathbf{x}-\mathbf{P}\mathbf{x}, \mathbf{x}-\mathbf{P}\mathbf{x}\rangle = \frac{1}{2}\|\mathbf{x}\|^2 - \operatorname{tr}(\mathbf{P}\mathbf{x}\mathbf{x}^\top) + \frac{1}{2}\operatorname{tr}(\mathbf{P}^\top\mathbf{P}\mathbf{x}\mathbf{x}^\top) \tag{1.8}$$

from which the simple[4] rules of differentiation

$$\nabla_{\mathbf{A}} \operatorname{tr}(\mathbf{A}\mathbf{B}) = \mathbf{B}^\top, \quad \nabla_{\mathbf{A}} \operatorname{tr}(\mathbf{A}^\top\mathbf{A}\mathbf{B}) = \mathbf{A}\mathbf{B}^\top + \mathbf{A}\mathbf{B},$$

yield

$$\nabla_{\mathbf{P}}\ell(\mathbf{x};\mathbf{P}) = (\mathbf{P}-\mathbf{I})\mathbf{x}\mathbf{x}^\top\,. \tag{1.9}$$

Furthermore by the chain rule for matrix derivatives[5]

$$\nabla_{\mathbf{W}}\ell(\mathbf{x};\mathbf{V}\mathbf{W}) = \mathbf{V}^\top(\mathbf{P}-\mathbf{I})\mathbf{x}\mathbf{x}^\top, \tag{1.10}$$

$$\nabla_{\mathbf{V}}\ell(\mathbf{x};\mathbf{V}\mathbf{W}) = (\mathbf{P}-\mathbf{I})\mathbf{x}\mathbf{x}^\top\mathbf{W}^\top\,. \tag{1.11}$$

Stochastic gradient descent (SGD) then picks a data point at random ($\sim \nu$ or uniformly from the sample) and evolves the iterates

$$\mathbf{W} \leftarrow \mathbf{W} - \eta\nabla_{\mathbf{W}}\ell(\mathbf{x};\mathbf{V}\mathbf{W}), \quad \mathbf{V} \leftarrow \mathbf{V} - \eta\nabla_{\mathbf{V}}\ell(\mathbf{x};\mathbf{V}\mathbf{W}), \quad \eta > 0\,. \tag{1.12}$$

We will not get into the topic of analyzing gradient descent or SGD at this point. We conclude by showing some code for batch gradient descent. Note that we can reduce the data to its sufficient statistics beforehand, making the computation vastly more efficient if the data dimensionality is smaller than the sample size.

Example 1.5 We can directly re-write the above equations in NumPy, replacing $\mathbf{x}\mathbf{x}^\top$ with the covariance matrix.

```
Delta = (V@W-np.identity(n)) @ covar
gradV = Delta @ np.transpose(W)
gradW = np.transpose(V) @ Delta
V     = V - eta * gradV
W     = W - eta * gradW
```

■

→ Gradients for weight matrices can easily be derived and used as the basis for (stochastic) gradient descent. This can leverage tools from Deep Learning.

Exercise 1.4 Implement parameter sharing via $\mathbf{W} = \mathbf{V}^\top$ and modify the gradient-based learning. Is there an advantage? Investigate this experimentally. ■

[4] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

[5] Or, alternatively, the cyclic property and the formula $\nabla_{\mathbf{A}} \operatorname{tr}(\mathbf{A}^\top\mathbf{B}\mathbf{A}\mathbf{C}) = \mathbf{B}\mathbf{A}\mathbf{C} + \mathbf{B}^\top\mathbf{A}\mathbf{C}^\top$.

Exercise 1.5 Implement a version of the linear autoencoder that extracts the PCA basis. How can you modify the objective and how does that change the gradient? Visualize the eigenimages. ■

1.6 Non-Linear Autoencoder

Do Add some example project and code to train deep autoencoders, e.g. Fahsion-MNIST[6]. Show simple use of PyTorch and vision libraries. Benchmark relative to linear autoencoder.

1.7 Conclusion

1. It is important to reflect on the model structure, e.g. identifiability, as well as possible data preprocessing.
2. The choice of objective function often has a profound influence on the solution, e.g. the squared error is directly tied to (orthogonal) projections.
3. It is often helpful to characterize the solution mathematically. Where this is not possible for the model under consideration, simplifications should be considered.
4. Mathematical insights also directly inform the choice of learning algorithm. There is a trade-off then between specialized methods and generic ones.

[6] `https://debuggercafe.com/implementing-deep-autoencoder-in-pytorch/`

2. Matrix Approximation

In this chapter we study the fundamental problem of *matrix approximation*. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, there are typically two related types of questions. First, we assume that a subset of entries is given exactly and we need to fill in the missing entries – this is commonly known as matrix completion or matrix reconstruction. Second, we assume to observe noisy entries of a matrix and want to infer the underlying matrix based on some *a priori* assumptions on the ground truth. In both cases, low rank matrices play a fundamental role and so do approximate factorizations of $\mathbf{A}$ into a product of matrices $\mathbf{A} \approx \mathbf{UV}$ with small inner dimension.

2.1 Collaborative Filtering

2.1.1 Matrices with Interdependent Entries

We are interested in the following problem: Assume that we are given a (sparse) matrix $\mathbf{A}$ for which a fraction of the entries has been observed, can we fill in the missing entries and reconstruct the underlying matrix?

$$\mathbf{A} = \begin{pmatrix} 3 & ? & ? & 2 \\ 0 & \frac{1}{2} & ? & ? \\ ? & ? & -2 & -\frac{4}{5} \end{pmatrix} \overset{\text{completion}}{\Longrightarrow} \quad ? \tag{2.1}$$

Without further specification, the problem is obviously ill-posed: if entries are generated independently at random, then there is no information in observed entries about unobserved ones. However, in case of practical interest there will be dependencies between entries. For instance, if the rows and column of the matrix are *meaningful*, then we expect (at least) the entries within the same column or the same row *not* to be independent. Hence they will carry information about each other, which may allow reconstructing or approximating missing entries.

→ A minimal assumption in matrix reconstruction is that entries within the same row or the same column are not independent.

What are the implications of such a minimal dependency assumption? Are entries that neither share a row, nor or column, “independent" of each other and in which sense? To add precision, let us treat matrix entries as random variables (i.e. noisy measurements). The minimal dependence assumption can then be described via conditional independence:

$$a_{ij} \perp\!\!\!\perp \{a_{kl} : k \neq i \;\wedge\; l \neq j\} \mid \{a_{il} : l \neq j\} \cup \{a_{kj} : k \neq i\} \tag{2.2}$$

In plain English: conditioning on all entries in the row and column of an entry a_{ij} renders it independent of the rest of the entries. However note that it immediately follows from the rules of probability that *indirect* dependencies between all entries emerge, if we do not observe all entries in the corresponding row and column. In particular, we may not assume that any two entries are marginally independent $a_{ij} \not\perp\!\!\!\perp a_{kl}$ for any ij, kl. Such non-trivial dependency structures make matrix completion and reconstructing a challenging problem.

→ Even under restrictive conditional independence assumptions, whenever a matrix is only partially observed, it effectively creates (indirect) couplings between all entries.

2.1.2 Recommender Systems

There is a classical problem in the area of user preference modeling, which is closely tied to the development of *recommender systems*. In such systems, the goal is to be able to recommend or pre-select relevant items in a personalized manner, based on a person's history or profile. Recommender systems have been invented in the early 1990s and are today in widespread use on the internet, powering applications, for instance, in e-commerce and social media, where ratings are solicited for movies, news, restaurants, products etc. We will abstract away from the specific domain and assume that people rate *items* on some ordinal rating scale, e.g. the popular 1-5 star scale or numerical rating from 1-10. We will use the convention that columns of the rating matrix corresponds to items and the rows to people or *users*.

→ In recommender systems, the matrix to be completed consists of ratings provided by users (rows) about items (columns).

As an example data set, we make use of the Movielens data set.

The typical challenge with such data sets is that ratings are only partially known, leading to highly sparse data. For instance, the famous Netflix Prize data set contains over 100M ratings solicited from 480k people, which means there are roughly 200 ratings per person (which is quite a lot), however, there are more than 17k movies, leading to a sparsity of about 1%. In many applications, sparsity can be much lower than that and easily dropping to the per mille range and below.

2.1.3 Collaborative vs. Content-Based Filtering

In designing recommender systems, one distinguishes *collaborative filtering* and *content-based filtering*. In the later case, one aims to build feature-based predictors, possibly in a custom manner for each person. However, as the per user data is scarce, this is often infeasibility. In addition, it may be difficult to extract predictive features from items such as music, images, or video. The key idea of collaborative filtering is to instead exploit the similarity between people's ratings to learn from the collective data provided by the community or customer base as a whole. Note that in practice, one would typically aim to exploit both sources of information and design hybrid systems.

There are also other relevant distinctions, most importantly the one between explicit preference solicitations (like described above) and implicit preferences. In the later case, users are typically not asked to provide feedback, but reveal preferential choices in their actions, for instance, by choosing one item over other alternatives (click, purchase, view, etc.). Here we will focus solely on explicit preference ratings.

2.1.4 Preprocessing

There is an often neglected aspect that we want to briefly point out, which is the question of data preprocessing. We always have to ask, what assumptions a model is making about the data, for instance, its measurement scales or about possible invariances? Specifically, in the case of ratings, we may want to consider shift and scale dependencies, also in a person- or item-specific manner. Let the rating matrix by given by $\mathbf{A} \in \mathbb{R}^{n \times m}$, where n is the number of people and m the number of items. We can compute mean ratings per item or person, writing in matrix form

$$\boldsymbol{\mu}^{\text{row}} = \frac{1}{m}\mathbf{A}\mathbf{1}_m \in \mathbb{R}^n, \quad \boldsymbol{\mu}^{\text{col}} = \frac{1}{n}\mathbf{A}^\top \mathbf{1}_n \in \mathbb{R}^m \tag{2.3}$$

and then normalize either way

$$\mathbf{A} \xleftarrow{\text{rows}} \mathbf{A}\left[\mathbf{I}_{m\times m} - \frac{1}{m}\mathbf{1}_m\mathbf{1}_m^\top\right], \quad \mathbf{A} \xleftarrow{\text{cols}} \left[\mathbf{I}_{n\times n} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top\right]\mathbf{A} \tag{2.4}$$

Exercise 2.1 Verify that the above matrix formulas indeed normalize rows and cols, respectively. ■

However, the formulae for the row and column means are only true in the fully observed case. In the partially observed case, we introduce an observation matrix

$$\Omega \in \{0,1\}^{n\times m} : \ \omega_{ij} = 1 \iff a_{ij} \text{ observed} \tag{2.5}$$

such that we can define the average ratings

$$\mu_i^{\text{row}} = \frac{\sum_j \omega_{ij} a_{ij}}{\max\{1, \sum_j \omega_{ij}\}}, \quad \mu_j^{\text{col}} = \frac{\sum_i \omega_{ij} a_{ij}}{\max\{1, \sum_i \omega_{ij}\}}, \tag{2.6}$$

and then center ratings by

$$a_{ij} \xleftarrow{\text{row}} a_{ij} - \mu_i^{\text{row}} \quad \text{or} \quad a_{ij} \xleftarrow{\text{col}} a_{ij} - \mu_j^{\text{col}} \tag{2.7}$$

Sometimes it may also make sense to normalize the variance to 1. Again, this can be done in a per row (item) or column (people) manner.

Definition 2.1.1 Let X be a score with $\mathbf{E}[X] = \mu$ and $\mathbf{Var}[X] = \sigma^2$, then its standardized score (or z-score) is given by

$$Z = \frac{X - \mu}{\sigma}$$

As many matrix models (as discussed below) may not be invariant to such transformations, one has to use domain knowledge and/or use some degree of experimentation.

R In the early collaborative filtering systems in the 1990s it was believed to be most natural to z-score normalize ratings per user or person. This way, one would account for certain users being generally more positive or negative or make different use of the dynamic range of the rating scale. However it was then a 'research discovery' (see [39], highly cited) to find that normalizing items (i.e. the transposed view) was more effective.

Exercise 2.2 Take the MovieLens data set and mean-normalize (a) columns, (b) rows to zero. Calculate a prediction for missing values and compare the mean squared error of (a) vs. (b) on the test data, where the column and row means, respectively, are used to fill-in the missing ratings. ■

2.2 Rank 1 Model

We will start with a simple rank 1 model that will be analyzed and discussed in detail to build-up intuitions. This also highlights the importance to start simple to pave the way for more complex and sophisticated models.

Q What is the simplest – yet interesting – matrix model that couples entries in each row and each column?

2.2.1 Outer Product Model

Our starting point will be a simple model that associates a single parameter with each row and column (cf. discussion above). It seems natural to consider a bi-linear model of matrices

$$\mathbf{A} \approx \mathbf{u}\mathbf{v}^\top, \quad \mathbf{u} \in \mathbb{R}^n, \quad \mathbf{v} \in \mathbb{R}^m . \tag{2.8}$$

In this model we approximate each entry by a product $a_{ij} \approx u_i v_j$, where a scalar parameter u_i is associated with the i-th row and v_j with the j-th column. Note that we have $n + m$ parameters to determine a total of nm matrix entries, which makes the model parsimonious. There is an obvious non-identifiability in the model as we can scale $\mathbf{u}$ by some $\alpha \in \mathbb{R} - \{0\}$ and simulatenously $\mathbf{v}$ by $1/\alpha$ to leave their outer product invariant. Using the squared error as a loss function then results in the optimization problem

$$\mathbf{u}, \mathbf{v} \to \min \ell(\mathbf{u}, \mathbf{v}) := \frac{1}{2} \|\Pi_\Omega(\mathbf{A} - \mathbf{u}\mathbf{v}^\top)\|_F^2 . \tag{2.9}$$

Here the projection to $\mathbf{\Omega}$ is defined as

$$\Pi_\Omega(\mathbf{R}) = \mathbf{R} \odot \mathbf{\Omega}, \quad \odot\text{: Hadamard product} \tag{2.10}$$

R By the definition of the Frobenius norm, this can be equivalently written as

$$\ell(\mathbf{u}, \mathbf{v}) = \frac{1}{2} \sum_{i,j} \omega_{ij} (a_{ij} - u_i v_j)^2 \,.$$

The matrix notation will turn out to be helpful subsequently.

R Note that $\mathbf{u}, \mathbf{v}$ are determined solely based on the observed values, however they extrapolate to a full matrix. It is clear that if there are rows or columns without any observations, the corresponding parameters cannot be identified (and are thus arbitrary)

Proposition 2.2.1 Every rank 1 matrix can be represented as an outer product of two vectors and every pair of vectors defines a rank 1 matrix via their outer product.

→ The outer-product model approximate a matrix over the space of rank one matrices.

2.2.2 Scalar Case

To gain insights into a model it is often illuminating to start with the simplest possible special case and see what properties emerge. In the above case, let us look at the scalar problem,

$$\ell(u, v) = \frac{1}{2}(a - uv)^2 \tag{2.11}$$

We want to find two numbers whose product equals a third. Let us look at the gradient field induced by the squared error, which for $a = 1$ is depicted in Figure 2.1. By the chain rule

$$\frac{\partial \ell}{\partial u} = \delta v, \quad \frac{\partial \ell}{\partial v} = \delta u, \quad \delta := uv - a \tag{2.12}$$

It is clear that the gradient is zero (red lines) on a hyperbola with two branches. Performing gradient descent will – dependent on the starting point – lead to some point on the hyperbola. What happens at the origin? There is also an isolated critical point at the origin, which can be seen to be a saddle point as

$$\nabla^2 \ell(u, v) = \begin{pmatrix} v^2 & 2uv - a \\ 2uv - a & u^2 \end{pmatrix}, \quad \nabla^2 \ell(0, 0) = \begin{pmatrix} 0 & -a \\ -a & 0 \end{pmatrix} \tag{2.13}$$

and the characteristic polynomial of the later is

$$\det(\lambda \mathbf{I} - \nabla^2 \ell(0, 0)) = \lambda^2 - a^2 = (\lambda - a)(\lambda + a) \tag{2.14}$$

which means the Hessian matrix is indefinite at the origin as it has eigenvalues $-a$ and a. Unless $a = 0$, the saddle point always has an escape direction (i.e. a direction of negative curvature).

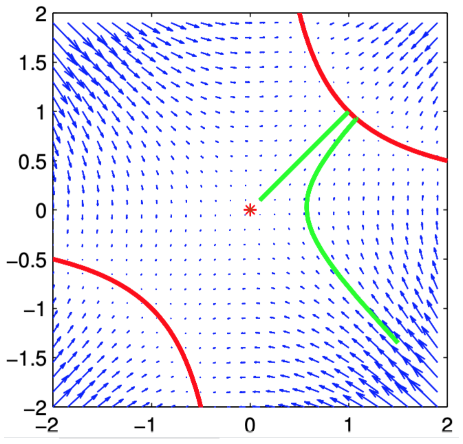


Figure 2.1: Gradient flow of $\frac{1}{2}(1-uv)^2$. Zero loss hyperbola in red, exemplary gradient flow trajectories in green.

$\rightarrow$ The critical point set of $\ell = \frac{1}{2}(a-uv)^2$ is a minimizing hyperbola as well as a saddle point at the origin, which is unstable under gradient dynamics.

2.2.3 Convexity

Let us connect this observation to the important concept of convexity.

Definition 2.2.1 A set in a vector space or affine space is convex if the line segment connecting any two points in the set is also in the set.

Definition 2.2.2 A function f is convex over a domain R, if for all $x, y \in R$

$$f(tx + (1-t)y) \le tf(x) + (1-t)f(y), \quad \forall t \in [0;1]$$

Proposition 2.2.2 If f is differentiable, convexity is equivalent to the condition

$$f(x) \ge f(y) + \nabla f(y) \cdot (x-y), \quad \forall x, y \in R$$

Proposition 2.2.3 If f is twice differentiable, convexity on a convex domain R is equivalent to the condition

$$\nabla^2 f(x) \succeq \mathbf{0}, \quad \forall x \in \mathrm{Int}(R)$$

We see that the objective in (2.11) is non-convex except for the special choice of $a = 0$, which follows from the fact that a saddle point exists at the origin. Convexity is a very fundamental property and it is an important demarcation line in optimization between tractable and (possibly) intractable problems. As we will see, matrix completion is a problem that while being non-convex can be analyzed in-depth, approximated well in practice and solved exactly for some special cases.

Going back to the general case, let us calculate the gradient. We can do this elementary via the partial derivatives, which directly generalizes the scalar case

$$\frac{\partial \ell}{\partial u_i} = \sum_j (u_i v_j - a_{ij}) v_j, \quad \frac{\partial \ell}{\partial v_j} = \sum_i (u_i v_j - a_{ij}) u_i \,. \tag{2.15}$$

It is often more compact to write it in vector/matrix notation.

$$\nabla_{\mathbf{u}}\ell = \mathbf{R}\mathbf{v}, \quad \nabla_{\mathbf{v}}\ell = \mathbf{R}^\top \mathbf{u}, \quad \mathbf{R} := (\mathbf{u}\mathbf{v}^\top - \mathbf{A}) \tag{2.16}$$

We can derive this equation either by re-vectorizing the formula for the partial derivatives or by using multivariate calculus with the Frobenius norm

$$\ell(\mathbf{u},\mathbf{v}) = \tfrac{1}{2}\|\mathbf{R}(\mathbf{u},\mathbf{v})\|_F^2, \quad \nabla_{\mathbf{R}}\tfrac{1}{2}\|\mathbf{R}\|_F^2 = \mathbf{R}\,. \tag{2.17}$$

Exercise 2.3 Use the above formula to derive the gradients. ■

As far as convexity is concerned, let us also derive the Hessian for the multivariate case

$$\nabla^2\ell(\mathbf{u},\mathbf{v}) = \begin{pmatrix} \|\mathbf{v}\|^2\mathbf{I}_{n\times n} & 2\mathbf{u}\mathbf{v}^\top - \mathbf{A} \\ (2\mathbf{u}\mathbf{v}^\top - \mathbf{A})^\top & \|\mathbf{u}\|^2\mathbf{I}_{m\times m} \end{pmatrix}, \quad \nabla^2\ell(\mathbf{0},\mathbf{0}) = \begin{pmatrix} \mathbf{0} & -\mathbf{A} \\ -\mathbf{A}^\top & \mathbf{0} \end{pmatrix} \tag{2.18}$$

We see that the step from the simple scalar case to the multidimensional case involves more abstract vector calculus, but is essentially small. Again, the Hessian at zero is not positive semi-definite, unless $\mathbf{A}$ is nilpotent (has all zero eigenvalues). The problem is non-convex in all dimensions m, n.

→ The outer product matrix reconstruction objective is generally non-convex.

2.2.4 Gradient Dynamics

We will investigate different algorithms for the matrix reconstruction problems below. But right now, let us try to get an understanding of the gradient dynamics, if we were to iteratively minimize the objective via gradient descent. We use this as an illustration of ODEs (ordinary differential equations) to understand gradient flows (i.e. the dynamics of gradient descent in the limit of small step sizes) and will focus on the case $n = m = 1$. Assume (as a simplification) that $a > 0$ and we initialize $u = v = c \in \mathbb{R}$, typically to a small (but non-zero – remember the saddle point!) value. This case is easier as u and v evolve in-synch and we can study the evolution of $x = uv = u^2$. First of all, note that the negative gradient flow relates the rate of change of a variable to its negative derivative:

$$\frac{du}{dt} = -v(uv-a), \quad \frac{dx}{dt} = \frac{du^2}{dt} = -2uv(uv-a) = -2x(x-a) \tag{2.19}$$

We can now solve this ODE manually or make use of a symbolic solver like Mathematica

```
Dsolve[x'(t) = 2ax(t)-2x(t)^2,x(0)==c^2]
```

This yields the analytic solution

$$x(t) = \frac{ae^{2at}}{e^{2at} - 1 + a/c^2} = a + \frac{ac^2 - a^2}{c^2e^{2at} + a - c^2} \tag{2.20}$$

from which we can determine how long it takes to get ϵ close to the target a.

Exercise 2.4 Solve $x(t) = |a - \epsilon|$ for t to determine how long it takes to get ϵ close to the optimum at $x = a$ ■

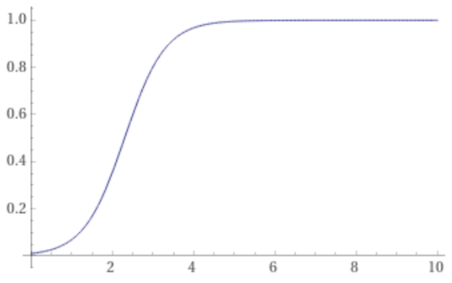


Figure 2.2: Solution of the ODE for $a = 1$ and starting point $c = 0.01$.

→ The negative gradient flow converges to a solution on the hyperbola, if initialized $(u(0), v(0)) \neq (0, 0)$ (shown for the special case $u(0) = v(0)$, true in general). The solution can be analytically calculated.

So in summary, we see that although the objective is non-convex, in special cases even simple first order methods like gradient descent will find the global optimum (here only shown in the continuous time limit, revisited later). Reflecting on the analyses pursued above, we want to stress again the usefulness of simple back-of-the-envelop calculations to gain insights into qualitative (convexity, critical points) as well as quantitative aspects of the problem (gradient flow dynamics).

2.2.5 Fully Observed Matrix

Let us conclude our discussion of the rank 1 model by considering the case where *all* matrix elements are observed. This setting is sensible, if we knew that the underlying matrix is of rank 1, yet entries have been corrupted (iid by additive Gaussian noise). We will make use of a little trick that is often helpful.

Proposition 2.2.4 $\|\mathbf{R}\|_F^2 = \mathrm{tr}(\mathbf{R}^\top \mathbf{R})$

Proof.

$$\mathrm{tr}(\mathbf{R}^\top \mathbf{R}) = \sum_i \Big[\sum_j r_{ij} r_{ij} \Big] = \sum_{i,j} r_{ij}^2 = \|\mathbf{R}\|_F^2$$

■

This allows us to rewrite the objective function

$$\ell(\mathbf{u}, \mathbf{v}) = \frac{1}{2}\|\mathbf{A} - \mathbf{u}\mathbf{v}^\top\|_F^2 = \frac{1}{2}\mathrm{tr}(\mathbf{A}^\top\mathbf{A} - \mathbf{A}^\top\mathbf{u}\mathbf{v}^\top - \mathbf{v}\mathbf{u}^\top\mathbf{A} + \mathbf{v}\mathbf{u}^\top\mathbf{u}\mathbf{v}^\top) \tag{2.21}$$

Because of the linearity of the trace, we can break this up into four trace terms, of which the first is constant (and can be dropped). Moreover, as the trace is invariant under cyclic permutations of matrices, we get two relevant terms

$$\ell(\mathbf{u}, \mathbf{v}) = \mathrm{const} + \frac{1}{2}\mathrm{tr}(\mathbf{u}^\top\mathbf{u}\,\mathbf{v}^\top\mathbf{v}) - \mathrm{tr}(\mathbf{u}^\top\mathbf{A}\mathbf{v}) = \frac{1}{2}\|\mathbf{u}\|^2\|\mathbf{v}\|^2 - \mathbf{u}^\top\mathbf{A}\mathbf{v} \tag{2.22}$$

The directionality of $\mathbf{u}, \mathbf{v}$ is purely determined by the last term, so we can solve for unit vectors (and then rescale)

$$(\mathbf{u}, \mathbf{v}) \longrightarrow \max\{\mathbf{u}^\top\mathbf{A}\mathbf{v}\}, \quad \text{s.t.} \quad \|\mathbf{u}\| = \|\mathbf{v}\| = 1 \tag{2.23}$$

How do we solve such a problem? We use Lagrange multipliers and write

$$\mathcal{L} = \mathbf{u}^\top \mathbf{A}\mathbf{v} - \lambda\, \mathbf{u}\cdot\mathbf{u} - \mu\, \mathbf{v}\cdot\mathbf{v} \tag{2.24}$$

which yields

$$\nabla_{\mathbf{u}}\mathcal{L} = \mathbf{A}\mathbf{v} - 2\lambda\mathbf{u} \overset{!}{=} \mathbf{0} \Longrightarrow \mathbf{u} = \frac{\mathbf{A}\mathbf{v}}{\|\mathbf{A}\mathbf{v}\|}, \quad [...] \Longrightarrow \mathbf{v} = \frac{\mathbf{A}^\top\mathbf{u}}{\|\mathbf{A}^\top\mathbf{u}\|} \tag{2.25}$$

If we put these two optimality conditions together, we get the eigenvector equations

$$\mathbf{u} \propto (\mathbf{A}\mathbf{A}^\top)\mathbf{u}, \quad \mathbf{v} \propto (\mathbf{A}^\top\mathbf{A})\mathbf{v} \tag{2.26}$$

As we maximize the objective, this implies that $\mathbf{u}$ should be proportional to the principal eigenvector of the matrix $\mathbf{A}\mathbf{A}^\top$, whereas $\mathbf{v}$ should be proportional to the principal eigenvector of $\mathbf{A}^\top\mathbf{A}$. Note that we can compute both simultaneously via power iterations, which computes $\mathbf{u}$ and $\mathbf{v}$ in alternation via (2.25).

Exercise 2.5 Show that the scaling factor of $\mathbf{u}^*$ can be recovered from the unit vector $\mathbf{u}$ via $\mathbf{A}\mathbf{v}$ (whereas we can leave $\mathbf{v}^* = \mathbf{v}$ a unit vector – why?). ■

We will generalize this result in the context of the Singular Value Decomposition (SVD).

2.3 Singular Value Decomposition

2.3.1 Definition & Properties

The Singular Value Decomposition (SVD) is an indispensable tool for algorithms and analysis alike. Let us first provide a precise formal definition.

Theorem 2.3.1 For each matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, there exists orthogonal matrices $\mathbf{U} \in \mathbb{R}^{n\times n}$ and $\mathbf{V} \in \mathbb{R}^{m\times m}$ such that $\mathbf{A}$ can be expressed as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top, \quad \mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \dots, \sigma_{\min\{n,m\}}), \quad \sigma_i \ge \sigma_{i+1}\ (\forall i)$$

Here a rectangular diagonal matrix is a matrix with entries on the main diagonal, padded with zeros. The set of singular values in the SVD are unique, however the left/right singular vectors (columns or $\mathbf{U}$ and $\mathbf{V}$) can have arbitrary sign. In the case of multiplicities of singular values, corresponding subspaces (but not their basis vectors) are unique.

We will first show how the SVD relates to matrix norms.

Proposition 2.3.2 Let the SVD of $\mathbf{A} \in \mathbb{R}^{n\times m}$ be given by $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, then

$$\|\mathbf{A}\|_F^2 = \sum_{i=1}^{\min\{n,m\}} \sigma_i^2$$

Proof. Making use of the properties of the trace

$$\mathrm{tr}(\mathbf{A}^\top\mathbf{A}) = \mathrm{tr}(\mathbf{V}\mathbf{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top) = \mathrm{tr}(\mathbf{V}\mathbf{\Sigma}^\top\mathbf{\Sigma}\mathbf{V}^\top) = \mathrm{tr}(\mathbf{\Sigma}^\top\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}) = \mathrm{tr}(\mathbf{\Sigma}^\top\mathbf{\Sigma})$$

■

Proposition 2.3.3 Let the SVD of $\mathbf{A} \in \mathbb{R}^{n \times m}$ be given by $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, then

$$\|\mathbf{A}\|_2 := \sup\{\|\mathbf{A}\mathbf{x}\| : \|\mathbf{x}\| = 1\} = \sigma_1$$

Proof. Orthogonal matrices preserve the Euclidean norm

$$\sup\{\|\mathbf{A}\mathbf{x}\|_2 : \|\mathbf{x}\| = 1\} = \sup\{\|\mathbf{\Sigma}\mathbf{V}^\top\mathbf{x}\| : \|\mathbf{x}\| = 1\} = \sup\{\|\mathbf{\Sigma}\mathbf{z}\| : \|\mathbf{z}\| = 1\} = \|\mathbf{\Sigma}\|_2$$

Noting that the 2-norm of a diagonal matrix is given by the largest absolute value of its elements. ■

2.3.2 SVD and Low-Rank Approximations

The following theorem is fundamental to many low-rank approximation problems. It states that by pruning the singular values below σ_k in the SVD representation, we get an optimal rank k approximation of a matrix. This means that approximations for any k can directly be read-off the SVD.

Theorem 2.3.4 — Eckart-Young. Given $\mathbf{A} \in \mathbb{R}^{n \times m}$ with SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. Then for each $1 \le k \le \min\{n, m\}$

$$\mathbf{A}_k := \mathbf{U}\,\mathrm{diag}(\sigma_1, \dots, \sigma_k)\mathbf{V}^\top \in \arg\min\{\|\mathbf{A} - \mathbf{B}\|_F : \mathrm{rank}(\mathbf{B}) \le k\}\,.$$

Corollary 2.3.5 The squared error of low rank approximations can be expressed as

$$\|\mathbf{A} - \mathbf{A}_k\|_F^2 = \sum_{i=k+1}^{\mathrm{rank}(\mathbf{A})} \sigma_i^2$$

Proof. Follows easily from Proposition 2.3.2 as

$$\mathbf{A} - \mathbf{A}_k = \mathbf{U}\,\mathrm{diag}(0, \dots, 0, \sigma_{k+1}, \dots, \sigma_{\min\{n,m\}})\mathbf{V}^\top\,.$$

■

It is clear that by pruning the singular values, one can also prune respective columns of $\mathbf{U}$ and $\mathbf{V}$, creating what is know as a reduced SVD representation, which is more compact for computations. It turns out that the optimal low-rank approximation is also the best in terms of spectral norm

Proposition 2.3.6 The matrix $\mathbf{A}_k$ as defined in the theorem above fulfills

$$\mathbf{A}_k \in \arg\min\{\|\mathbf{A} - \mathbf{B}\|_2 : \ \mathrm{rank}(\mathbf{B}) \le k\}\,.$$

2.3.3 SVD and PCA

The SVD is intimiately related to eigendecomposition. First note that if $\mathbf{A}$ is square symmetric, then $\mathbf{U}$ and $\mathbf{V}$ have equal columns up to possible sign differences. Note that all eigenvalues of a symmetric matrix are real (but can be negative), yet the singluar values are always non-negative. If $\mathbf{A}$ is also positive semi-definite, then the SVD is equal to the eigendecomposition. The more interesting connection to spectral decompositions is to consider the "squares" of $\mathbf{A}$, namely $\mathbf{A}\mathbf{A}^\top \in \mathbb{R}^{n \times n}$ as well as $\mathbf{A}^\top\mathbf{A} \in \mathbb{R}^{m \times m}$. Here we see that by simply writing out

$$\mathbf{A}\mathbf{A}^\top = \mathbf{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top\mathbf{U}^\top = \mathbf{U}\,\mathrm{diag}(\sigma_1^2,\dots,\sigma_n^2)\mathbf{U}^\top, \tag{2.27}$$

$$\mathbf{A}^\top\mathbf{A} = \mathbf{V}^\top\boldsymbol{\Sigma}^\top\boldsymbol{\Sigma}\mathbf{V} = \mathbf{V}\,\mathrm{diag}(\sigma_1^2,\dots,\sigma_m^2)\mathbf{V}^\top, \tag{2.28}$$

where we have used the convention $\sigma_r = 0$ for $\min\{n,m\} < r \leq \max\{n,m\}$.

How does this apply to PCA? Remember that we need the eigendecomposition of the covariance matrix $\mathbf{E}[\mathbf{x}\mathbf{x}^\top]$, which in the case of a finite sample is $\frac{1}{s}\mathbf{X}\mathbf{X}^\top$, where $\mathbf{X}$ is the data matrix, containing patterns as rows. In this case the SVD of $\mathbf{X}$ can directly give us the desired principal eigenvectors as the columns of the matrix $\mathbf{U}$.

→ SVD can be applied to the data matrix to identify the principal eigenvectors of the covariance matrix (PCA).

2.3.4 SVD for Matrix Completion

A natural generalization of the simple rank 1 model discussed above is to consider rank k approximations. Note that these are essentially (additive) superpositions of k rank 1 matrices. If the matrix entries are completely observed, then the solution is represented via the SVD and also computable as such. To point out the obvious: the k principal left singular vectors, paired with the respective right singular vector counterpart are exactly these vectors forming outer products.

$$\mathbf{A} \approx \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \tag{2.29}$$

Coming back to the discussion above, we re-iterate that the low-rank approximation problem is non-convex, which includes the completely observed case. Hence, as SVD is efficiently computable with $\mathbf{O}(\min\{nm^2, mn^2\})$ operations, this is an important example of a solvable non-convex problem.

→ In the completely observed case, we can use SVD to calculate the best low-rank approximation of a matrix.

SVD has also been used at times in the context of collaborative filtering. However, this means we need to explicitly or implicitly impute values for the missing entries, for instance by identifying them with zeros. This is problematic as the semantics of a missing value is different from an observed zero. So this requires at the very least a mean normalization of rows or columns in order to make the zero point more meaningful. Even so, this is not a recommended approach in general.

→ In the case of incomplete observations, SVD is in general not applicable directly to compute low-rank approximations.

2.3.5 NP Hardness

Low-rank matrix approximation with a weighted Frobenius norm is defined as follows:

$$\hat{\mathbf{A}}_k = \arg\min \Big\{ \sum_{i,j} w_{ij}(a_{ij} - b_{ij})^2, \ \text{rank}(\mathbf{B}) = k \Big\} \tag{2.30}$$

where typically the weights are positive $w_{ij} > 0$ or binary $w_{ij} = \omega_{ij} \in \{0,1\}$ (as in the case of partial observations). In both cases the problem has been shown to be NP- hard [15], even for $k = 1$.

→ Low rank matrix reconstruction is NP hard and one has – in general – to resort to approximation algorithms. The completely observed case is special.

2.4 Alternating Least Squares

2.4.1 Parameter Interactions and Separability

Let us work with the factored parameterization via $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times m}$ such that $\mathbf{A} \approx \mathbf{UV}$ and an approximation error described by the (non-convex) objective

$$\ell(\mathbf{U}, \mathbf{V}) = \frac{1}{2}\|\Pi_\Omega(\mathbf{A} - \mathbf{UV})\|_F^2 + \lambda(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2), \quad \lambda > 0 \tag{2.31}$$

The additional regularization of the factor matrices is somewhat optional, but increases numerical (and statistical) stability.

Q How do parameters interact?

It is important to understand interactions between parameters as it is closely linked to the structure of the problem and can often be exploited. First note that the objective is a polynomial of degree 4 in the parameters with the following monomials

$$\omega_{ij} u_{ir} v_{rj} u_{is} v_{sj} \quad 1 \le r, s \le k \tag{2.32}$$

$$\omega_{ij} u_{ir} v_{rj} \quad 1 \le r \le k \tag{2.33}$$

$$u_{ir}^2, \ v_{rj}^2 \quad \text{from the regularizer} \tag{2.34}$$

From this we can directly read-off an important property by observing that all higher order products only involve exactly one row index i of $\mathbf{U}$ and one column index j of $\mathbf{V}$.

→ The parameter dependencies in the non-convex objective for matrix factorization form a bipartite graph between the rows of $\mathbf{U}$ and the columns of $\mathbf{V}$.

2.4.2 Separable Least Squares Problems

Because of the dependence (or rather independence) structure of the parameters we can separate out the part of the objective that depends on a column $\mathbf{v}_j$ of $\mathbf{V}$ (and completely analogous a row $\mathbf{u}_i$ of $\mathbf{U}$) as follows

$$\ell_{\mathbf{U}}(\mathbf{v}_j) = \frac{1}{2}\mathbf{v}_j^\top \Big(\sum_i \omega_{ij} \mathbf{u}_i \mathbf{u}_i^\top + 2\lambda \mathbf{I} \Big) \mathbf{v}_j - \Big(\sum_i \omega_{ij} a_{ij} \mathbf{u}_i^\top \Big) \mathbf{v}_j \tag{2.35}$$

Now if we treat $\mathbf{U}$ as a fixed matrix (and not a set of parameters), we can easily solve for each $\mathbf{v}_j$ as this is just a least squares problem, which has the solution

$$\mathbf{v}_j^* = \Big(\sum_i \omega_{ij} \mathbf{u}_i \mathbf{u}_i^\top + 2\lambda \mathbf{I} \Big)^{-1} \Big(\sum_i \omega_{ij} a_{ij} \mathbf{u}_i \Big) \tag{2.36}$$

Note that this involves the inverse of a $k \times k$ matrix, which is usually small in comparison to n and m. The regularization ensures invertability. Such explicit formulae may not only offer insights, but also significant advantages over generic optimization methods like gradient descent.

Exercise 2.6 Derive the explicit equation for the solution shown above and comment on its existence and uniqueness. ■

→ The separable subproblems are least squares problems of dimension k.

2.4.3 Alternating Subspace Optimization

Q How can we exploit the separability structure of the problem for optimization?

The algorithmic idea to exploit the problem structure is simple and intuitive. Given a choice of $\mathbf{U}$, we can (efficiently) minimize each of the m subproblem for $\{\mathbf{v}_j\}$. Note that this can be done with a high degree of parallelism. Effectively, we are iteratively optimizing over the subspace of parameters $\mathbf{V}$ keeping $\mathbf{U}$ fixed. Then we transpose the perspective and keep $\mathbf{V}$ fixed, optimizing over each of the n subproblems related to rows $\{\mathbf{u}_i\}$. While gradient descent makes a small update to all parameters, here we make no update to one "half" of the parameters, yet chose the other "half" in an optimal way. At an abstract level the algorithm hence alternates the two steps

$$\mathbf{V}^{t+1} \leftarrow \arg\min_{\mathbf{V}} \ell(\mathbf{U}^t, \mathbf{V}), \quad \mathbf{U}^{t+1} \leftarrow \arg\min_{\mathbf{U}} \ell(\mathbf{U}, \mathbf{V}^{t+1}) \tag{2.37}$$

As both steps optimize the same objective (only with regard to a complementary set of parameters), it is clear that ℓ will be monotonically decreasing under such alternating updates. Since $\ell \geq 0$ is lower bounded, the ALS algorithm will converge towards a fixed point.

→ ALS is an iterative algorithm that monotonically improves the objective and that converges to a fixed point.

In general, we cannot expect the solution of ALS to be the global minimizer of the problem, unless in special cases. The termination condition of ALS does not imply that there cannot be mixed update directions, where progress is possible. However, note the gradient vanishes at a fixed point of ALS

$$\mathbf{V}^* = \arg\min_{\mathbf{V}} \ell(\mathbf{U}^*, \mathbf{V}), \quad \mathbf{U}^* = \arg\min_{\mathbf{U}} \ell(\mathbf{U}, \mathbf{V}^*) \quad \Longrightarrow \quad \nabla \ell(\mathbf{U}^*, \mathbf{V}^*) = \mathbf{0} \tag{2.38}$$

The latter is clear as if there was any non-zero partial derivative, then we could improve by optimizing over the respective subspace (either $\mathbf{U}$ or $\mathbf{V}$), which contradicts the fixed point condition of ALS. In fact we also know that the two diagonal blocks of the $(\mathbf{U}, \mathbf{V})$-partitioned Hessian are positive definite (we have shown above that these are least squares problems). This analysis shows that ALS may converge to a local minimum, but leaves open the possibility that it may lead to a saddle point.

2.4.4 Program Code

Python offers support for sparse matrices in different storage formats (optimised for different use cases) via the library `scipy`. Some simple program code to generate sparse matrix with random sparsity pattern and random 5 star ratings is shown below.

```
# generating some random sparse matrix
def rand5star(num): # 5 star scale
  return np.random.randint(1,6,size=num)
A = scipy.sparse.random(rows,cols,0.1,data_rvs=rand5star,format='csr')

print(A.A) =>
[[0. 0. 0. 0. 0. 4. 0. 0. 0. 3.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 5. 0. 2. 0. 0. 0. 0. 3. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In implementing ALS, the offset vectors can be computed via

```
B = U.transpose() @ A   # rank x cols
```

and for the quadratic part we can compute all relevant rank 1 matrices

```
Q = np.ndarray((rows,rank,rank))
for row in range(rows):
    Q[row] = U[row,:].transpose() @ U[row,:]
```

We can then update $\mathbf{V}$ as specified in equation (2.36) as follows:

```
for col in range(cols):
    sum_r1_mat = 0
    for row in range(rows):
        if A[row, col]:
            sum_r1_mat += Q[row, :, :]
    inv = np.linalg.inv(sum_r1_mat + lam * np.identity(rank))
    V[:, col] = inv @ B[:, col]
```

The update of $\mathbf{U}$ works analogously.

2.4.5 Practical Considerations

The academic perspective is often driven by clear objectives, e.g. quality guarantees of algorithms or their runtime analysis. However, in real-world use cases many more factors may be of relevance. We highlight here an important, but simple aspect of ALS, which is that it can be used to *incrementally extend* a model. What we mean by this is the following. Imagine a recommender system which makes use of ratings predicted by a factored model $\mathbf{A} \approx \mathbf{UV}$. Often, computing, verifying and deploying such a model involves considerable effort and resources. On the other hand, there may constantly be new users entering the system and new items becoming available. The real world does not stand still!

Q How can we incrementally update a model without having to go back to the original data?

The ALS perspective provides a very simple and efficient solution. Note that we can compute an additional column $\mathbf{v}_{m+1}$ of $\mathbf{V}$ (or row $\mathbf{u}_{n+1}$ of $\mathbf{U}$) by solving a least squares problem that only involves the parameter matrix $\mathbf{U}$ (or $\mathbf{V}$) and the newly observed entries. Similarly, we can also update such vectors, if additional matrix entries are observed. We can do this cheaply, if we refrain from further alternating iterations, which offers a trade-off between accuracy and computational complexity. We can (1) recompute without iterations, (2) update a model taking an existing model as starting point, or (3) train a model from scratch.

$\rightarrow$ The separable model structure allows for efficient approximate incremental updates per row or column that only requires local data.

2.5 Projection Algorithms

In gradient descent and ALS, we have parameterized our model as a product of two matrix factors $\mathbf{A} \approx \mathbf{UV}$, which allowed us to encode the rank constraint directly into the parameterization, i.e. the inner dimension of $\mathbf{U}$ and $\mathbf{V}$. While this is perhaps the most natural approach, it is not the only possible avenue. Here we will discuss algorithms that will operate in an unconstrained parameterization, yet enforce the rank constraint on the reconstruction explicitly.

2.5.1 Singular Value Projection

The first algorithm exploits the fact that the projection of a matrix to the set of rank k matrices can be done efficiently via SVD. Projected gradient descent can then be written as follows

$$\mathbf{A}^0 = \mathbf{0}, \quad \mathbf{A}^{t+1} = \left[\mathbf{A}^t + \eta\, \Pi_\Omega(\mathbf{A} - \mathbf{A}^t)\right]_k, \quad \eta > 0 \tag{2.39}$$

The notation $[\cdot]_k$ extends the previous notation to denote the best rank k approximation to the matrix in brackets. It corresponds to a projection to the set of rank k matrices and can be computed via SVD. Clearly the expression in brackets is just the negative gradient of the Frobenius norm of the residual on observed entries. This is a projected gradient method that is known as singular value projection [21]. This method will converge with a carefully chosen stepsize. A more detailed analysis is beyond the scope of our lecture. Computationally, such an approach is attractive, if the SVD computations in the inner loop are tractable, for instance if k is small.

$\rightarrow$ In Singular Value Projection, SVD is used in combination with gradient descent to perform a non-convex projection to the space of rank k matrices.

2.5.2 Nuclear Norm Relaxation

Another related idea is to use a convex relaxation of the rank constraint.

Q Can we find a convex set of matrices which contains all rank k matrices but not too many more? What is the tightest relaxation?

The answer to this question is the nuclear norm (sometimes also called trace norm):

Definition 2.5.1 The *nuclear norm* of a matrix $\mathbf{A}$ is the sum of its singular values, $\|\mathbf{A}\|_* = \sum_{i=1}^{\mathrm{rank}(\mathbf{A})} \sigma_i$.

It is elucidating to compare the nuclear norm and the Frobenius norm. Denote the vector of singular values of $\mathbf{A}$ by $\sigma(\mathbf{A})$, then:

Proposition 2.5.1

$$\|\mathbf{A}\|_F = \|\sigma(\mathbf{A})\|_2, \quad \|\mathbf{A}\|_* = \|\sigma(\mathbf{A})\|_1$$

Hence we see that the nuclear norm generalizes the idea to use the 1-norm to find sparse solutions. Sparseness with regard to singular values is equivocal with low rank.

Definition 2.5.2 The convex envelope of a function $f : R \to \mathbb{R}$ is the largest convex function g for which $g \leq f$ on R.

We have the following result due to [13].

Theorem 2.5.2 The convex envelope of $\mathrm{rank}(\mathbf{A})$ on $R = \{\mathbf{A} : \|\mathbf{A}\|_2 \leq 1\}$ is $\|\mathbf{A}\|_*$.

This theorem provides the motivation for using the nuclear norm as a convex surrogate function to replace the rank.

While conceptually compelling, what are computational implications of using the nuclear norm? A foundational result involving the SVD is due to [5].

Proposition 2.5.3 Let $\mathbf{A}$ with SVD $\mathbf{A} = \mathbf{U}\,\mathrm{diag}(\sigma_i)\mathbf{V}^\top$ be given. Then

$$\mathrm{shrink}_\tau(\mathbf{A}) := \arg\min_{\mathbf{B}} \frac{1}{2}\|\mathbf{A}-\mathbf{B}\|_F^2 + \tau\|\mathbf{B}\|_* = \mathbf{U}\,\mathrm{diag}(\sigma_i - \tau)_+\mathbf{V}^\top \tag{2.40}$$

Procedurally we have abstracted the nuclear norm minimization into an operator, which shrinks each singular value by τ, while clipping the result at zero. Note that this shows how the nuclear norm induces sparseness: all singular values below τ will be zeroed out and hence the rank of $\mathrm{shrink}_\tau(\mathbf{A})$ will be monotonically decreasing with τ. What can we do with this basic operator as a computational subroutine? One possible answer is provided by the following proposition [5].

Proposition 2.5.4 With a suitable step size schedule $\eta_t > 0$, consider the iterate sequence:

$$\mathbf{A}^0 = \mathbf{0}, \quad \mathbf{A}^{t+1} = \mathbf{A}^t + \eta_t \Pi_\Omega(\mathbf{A} - \mathrm{shrink}_\tau(\mathbf{A}^t)) \tag{2.41}$$

Then, the sequence $\mathrm{shrink}_\tau(\mathbf{A}^t)$ will converge to

$$\mathrm{shrink}_\tau(\mathbf{A}^t) \overset{t\to\infty}{\longrightarrow} \mathbf{A}^* = \arg\min_{B} \left\{\|\mathbf{B}\|_* + \frac{1}{2\tau}\|\mathbf{B}\|_F^2\right\}, \quad \text{s.t.} \quad \Pi_\Omega(\mathbf{A}-\mathbf{B}) = \mathbf{0}$$

Note that the nuclear norm is the convex envelope of the rank function, as long as we can also control the Frobenius norm of possible solutions. This is why a choice of $\tau < \infty$ makes sense. The result will be exactly reproducing the observed entries, something that we cannot guarantee for the non-relaxed problem (irrespective of algorithm) as there may be no rank k matrix with a projected residual that is zero. The soft penalty on the hybrid of nuclear norm and Frobenius norm allows for that however.

In comparison to Singular Value Projection the convex relaxation approach does not

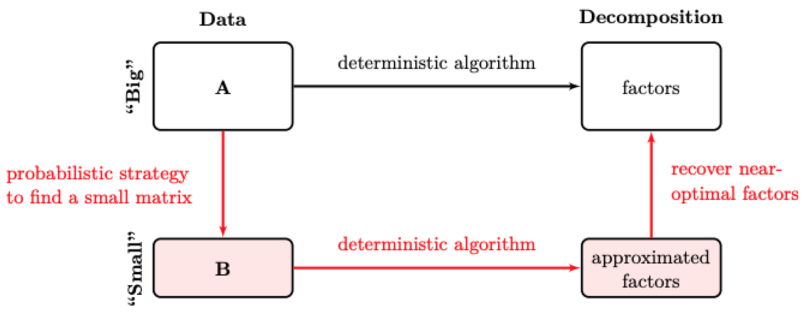


Figure 2.3: Schematic view of how to use random projections to compute matrix factorizations and decompositions. Taken from [18].

define a low rank sequence of matrices. Rather the SVD-based shrinkage operator is used implicitly to compute and update direction. Note that the iterates have the same sparseness structure as the data matrix $\mathbf{A}$, which can be a significant saving.

→ The convex relaxation approach to matrix completion maintains a sparse iterate sequence, whereas Singular Value projection maintains a rank k matrix. Both are beneficial relative to a naïve (i.e. dense) representation.

2.5.3 Randomized Algorithm for SVD

As we have seen, the algorithms presented in this section rely heavily on repeated SVD computations for projection or shrinkage. It should be clear that in practice one can tolerate some degree of inaccuracy for intermediate iterates.

Q Are there simple and scalable algorithms for SVD?

We therefore conclude this part by presenting a general and powerful use of randomized projections [18], which is also available in programming languages like R [12]. The basic idea applies to many types of matrix decompositions and is diagrammatically shown in Figure 2.3. Let us assume that (1) we can compute an orthogonal matrix $\mathbf{Q}$ with few(er) columns such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{A}$. Then we can (2) perform SVD of the smaller matrix $\mathbf{B} := \mathbf{Q}^\top \mathbf{A} = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^\top$ and (3) extend it to an approximate SVD for $\mathbf{A} \approx (\mathbf{Q}\tilde{\mathbf{U}})\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^\top$. Of course, in computing the SVD of $\mathbf{B}$, we can apply the same technique to its rows, thereby reducing both columns and rows of $\mathbf{A}$.

How can we find such a matrix $\mathbf{Q}$? A prototypical scheme is as follows. (1) Generate a random matrix with iid Gaussian entries $\mathbf{R} \in \mathbb{R}^{m \times 2k}$. (2) Calculate $\mathbf{Y} = (\mathbf{A}\mathbf{A}^\top)^q \mathbf{A}\mathbf{R}$ (e.g. for $q = 2$) by repeated matrix multiplications. (3) Construct an orthonormal basis for the image of $\mathbf{Y}$ (e.g. via Gram-Schmidt). This algorithm can be re-fined in many ways and was thoroughly analysed in the (excellent!) seminal paper [18].

→ The SVD can be efficiently computed via random projections, which also gives control on the trade-off of accuracy vs. complexity.

2.6 Exact Matrix Reconstruction

We have seen that the nuclear norm relaxation provides a systematic way to compute a low rank approximation of a matrix, which however, may not be the best approximation (in the Frobenius norm sense, say). Let us now consider the following question:

If the underlying matrix $\mathbf{A}$ is of rank k, are there conditions under which it can it be recovered exactly (with high probability)?

2.6.1 Degrees of Freedom

It is clear that a certain number S of matrix entries need to be observed in order to be able to even uniquely determine a rank k matrix. Counting these degrees of freedom provides a simple information theoretic bound. In the sequel, we will focus on quadratic matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$, which simplifies the analysis.

From the SVD of $\mathbf{A}$, we can read of the degrees of freedom as follows. We can chose k singular values. For the left and right singular vectors, the number of degrees of freedom is,

$$(n-1)+(n-2)+\cdots+(n-k) = nk - \sum_{i=1}^{k} i = nk - \frac{k(k+1)}{2}\,. \tag{2.42}$$

The reduction from nk is implied by the normalization and pairwise orthogonality constraints. So in total we get a necessary condition

$$S \geq k + 2\left(nk - \frac{k(k+1)}{2}\right) = 2nk - k^2 \tag{2.43}$$

From this one can conclude:

A rank k square matrix of dimension $n \times n$ is not reconstructable, if the number of observed matrix entries is $S < 2nk - k^2$.

From this analysis it is clear that at least $\mathbf{O}(nk)$ entries of $\mathbf{A}$ need to be sampled.

2.6.2 Coupon Collector

Let us naïvely consider a coupon collector argument (which can also be made more formally). If we need to collect N pieces of information through random sampling, we will often re-sample information that is already available. So there is an extra price to be paid for random selection. The simplest problem is known as the coupon collector's problem.

If items contain one of n coupon each (at random), what is the probability that more than t items need to be acquired to collect all n coupons? How many items need to be acquired on average?

Let us quickly present the argument. Denote by t_i the number of items that need to be acquired in order to acquire a new i-th coupon, given that $(i-1)$ have been acquired. t_i follows a geometric distribution with expectation

$$\mathbf{E}[t_i] = \frac{n}{n-i+1} \tag{2.44}$$

Then the time to acquire all coupons is in expectation

$$\mathbf{E}[T] = \sum_{i=1}^{n} \mathbf{E}[t_i] = n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = nH_n \tag{2.45}$$

where H_n is the n-th harmonic number, $H_n = \log n + \mathbf{O}(1)$.

→ In expectation, the coupon collector has to (asymptotically) over-sample items by a factor of $\log n$.

Informally applied to the matrix reconstruction problem, we have to expect an extra factor $\mathbf{O}(\log nk) = \mathbf{O}(\log n)$.

2.6.3 Incoherence

For matrix reconstruction to work, one requires some additional regularity condition on the matrix, which in compressed sensing is often called a measure of incoherence. The need for this can be easily illustrated by a simple (counter) example. Assume that in the SVD of $\mathbf{A}$, $\mathbf{u}_1 = \mathbf{e}_i$, $\mathbf{v}_1 = \mathbf{e}_j$, then in order to recover the singular value σ_1, one needs to sample a_{ij} as the other entries of $\mathbf{A}$ contain no information about a_{ij} (cf. motivating discussion).

→ Low-rank matrix reconstruction from sparse samples is not possible without further regularity conditions.

Intuitively, one needs information to be sufficiently distributed. In the above case one can follow the approach of [6] and define first the projection matrices to the column and row space of a rank k matrix with SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$

$$\mathbf{P} := \sum_{i=1}^{k} \mathbf{u}_i\mathbf{u}_i^\top, \quad \mathbf{Q} := \sum_{i=1}^{k} \mathbf{v}_i\mathbf{v}_i^\top, \quad \text{as well as} \quad \mathbf{E} = \sum_{i=1}^{k} \mathbf{u}_i\mathbf{v}_i^\top \tag{2.46}$$

Then [6] require incoherence conditions of the type

$$|p_{ij}|, |q_{ij}| \overset{i\neq j}{\leq} \frac{\mu\sqrt{k}}{n}, \quad \left|p_{ii} - \frac{k}{n}\right|, \quad \left|q_{ii} - \frac{k}{n}\right| \leq \frac{\mu\sqrt{k}}{n}, \quad |e_{ij}| \leq \frac{\mu\sqrt{k}}{n}. \tag{2.47}$$

It is a bit beyond the scope of these lectures to explain these conditions. The interested reader can consult [6] and the literature on incoherence in compressed sensing.

2.6.4 Reconstruction Theorem

We will now state and discuss a beautiful and far-reaching result on matrix reconstruction as developed in [6]. We state the quadratic case, the rectangular one is similar.

Theorem 2.6.1 Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a rank k matrix that is incoherent with some $\mu \geq 1$ and for which S samples have been observed at random. Then there is a universal constant C such that if $S \geq C\mu^2 nk(\log n)^6$, then with probability at least $1 - n^{-3}$, $\mathbf{A}$ fulfills

$$\mathbf{A} = \arg\min_{\mathbf{B}} \|\mathbf{B}\|_*, \quad \text{subject to} \quad \Pi_\Omega(\mathbf{B}) = \Pi_\Omega(\mathbf{A})$$

This states that there is a critical number of observations after which the solution to the nuclear norm relaxation will recover the correct matrix with high probability. Note that up to poly-logarithmic factors, the critical value scales with kn and in some sense is close to the information-theoretic limit. This says essentially:

→ If the matrix is recoverable from its sampled entries, then it can “essentially” be recovered via nuclear norm minimization.

3. Latent Variable Models

3.1 Probabilistic Clustering Models

3.1.1 Mixture Models

Assume we are given a data set of patterns $\{\mathbf{x}_t : t = 1, \ldots, s\}$. The conceptually simplest family of latent variable models associates a categorical (random) variable Z_t with each pattern. The latent information tags a pattern as a member of a group or class. For this reason, we can think of such latent class models in terms of probabilistic clustering models. In statistics, they are known as *mixture models*.

→ Mixture models are probabilistic clustering models that associate a categorical variable with each pattern.

Let us make this more formal. Under a common iid assumption, each latent class variable will follow a categorical law

$$Z_t \overset{\text{iid}}{\sim} \text{Categ}(\pi_1, \ldots, \pi_k), \quad \mathbb{P}(Z_t = z) = \pi_z \tag{3.1}$$

Here $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ is an unknown parameter vector that encodes the prior probabilities or proportions for each class. Note that we typically do not know beforehand what these classes *mean*, but we often hope to *discover* meaningful classes via clustering.

Following the latent variable philosophy, one will specify a class conditional distribution $p(\mathbf{x}|z)$ for each class. We can then formally obtain a model for the observables by summing out the latent variables (i.e. by marginalization)

$$p(\mathbf{x}, z) = \pi_z \, p(\mathbf{x}|z), \quad p(\mathbf{x}) = \sum_{z=1}^{k} p(\mathbf{x}, z) = \sum_{z=1}^{k} \pi_z \, p(\mathbf{x}|z) \tag{3.2}$$

Note that as $\boldsymbol{\pi} \geq 0$, $\sum \pi_z = 1$, we have that:

→ Mixture distributions are convex combinations of class-specific distributions.

Typically, the class conditional distributions will be independently parameterized with parameters $\boldsymbol{\theta}_z$. Learning a mixture model involves fitting these parameters along with the mixture proportions, in total we have a model parameter vector $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k)$. For given parameters $\boldsymbol{\theta}$ we can use Bayes' rule to calculate the posteriors

$$\mathbb{P}(Z = z|\mathbf{x}; \boldsymbol{\theta}) = \frac{\pi_z \, p(\mathbf{x}; \boldsymbol{\theta}_z)}{\sum_{\zeta=1}^{k} \pi_\zeta \, p(\mathbf{x}; \boldsymbol{\theta}_\zeta)} \tag{3.3}$$

We can interpret these posteriors as soft clustering weights or – more precisely – probabilistic cluster memberships.

→ Posterior latent class probabilities represent a probabilistic clustering.

3.1.2 Maximum Likelihood Estimation

How can we infer or learn the model parameters $\boldsymbol{\theta}$ from data? Foremost we need an inference principle.

Q What is a suitable inference principle for latent class models?

A very popular approach with strong theoretical foundations is the *maximum likelihood principle*. It uses a likelihood function as an objective. What is a likelihood function? It is the probability distribution (e.g. probability density or mass function) thought of as a function of the parameters, while fixing the observed outcomes (i.e. the data). It is convenient to work on the logarithmic scale and define the *log-likelihood*. In the iid setting one gets:

$$\ell(\boldsymbol{\theta}; \{\mathbf{x}_1, \ldots, \mathbf{x}_s\}) = \sum_{t=1}^{s} \ln p(\mathbf{x}_t; \boldsymbol{\theta}) = \sum_{t=1}^{s} \ln \sum_{z=1}^{k} \pi_z \, p(\mathbf{x}_t; \boldsymbol{\theta}_z) \tag{3.4}$$

We are then left with an optimization problem to find the maximum likelihood estimate (MLE)

$$\boldsymbol{\theta}^{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \{\mathbf{x}_1, \ldots, \mathbf{x}_s\}) \tag{3.5}$$

→ Maximum likelihood inference suggests to chose the model parameters which maximize the probability of the observed data.

This leads to the natural follow-up question:

Q How can we compute the MLE of a mixture model?

One classical and widely-applicable approach is the well-known Expectation-Maximization (or short: EM) algorithm.

3.1.3 Expectation Maximization Algorithm

The EM algorithm is a versatile tool for learning in latent variable models. We discuss it here in the context of mixture model, but will re-visit extensions below. The basic step in EM is to apply Jensen's inequality to lower bound a concave function or – equivalently – upper bound a convex function. Formally let ϕ be a concave function and X be a random variable, then

$$\phi(\mathbf{E}[X]) \geq \mathbf{E}[\phi(X)] \tag{Jensen's inequality}$$

In plain English: averaging first and then applying a concave function to the average cannot yield a value that is smaller than the average of the function values. It is interesting to note that Jensen's inequality applies irrespective of the law that governs X – it just relies on the concavity property of ϕ.

There is a simple (and frequently applied) trick of how to make use of Jensen's inequality. Let us develop this trick in the abstract and consider $g = \ln \sum_{z=1}^{k} f_z$, and define a k-atom random variable as follows

$$\mathbb{P}(X = x) = \sum_{z=1}^{k} q_z \mathbb{I}\left[x = \frac{f_z}{q_z}\right] \tag{3.6}$$

Here the positive weights q_z are chosen arbitrarily, but such that $\sum q_z = 1$. Note that the probabilities enter in the atomic values that X takes. This may look strange at first, but leads to the following:

$$\begin{aligned} g = \ln \sum_z f_z = \ln \sum_z \frac{q_z f_z}{q_z} = \ln \mathbf{E}[X] \\ \geq \mathbf{E}[\ln X] = \sum_z q_z \ln \frac{f_z}{q_z} = \sum_z q_z \ln f_z + H(\mathbf{q}) \end{aligned} \tag{3.7}$$

where H denotes the entropy. As we can see, we have converted a logarithm of a sum of terms into a q-expectation of the logarithmic terms. This is known as a variational inequality as we have not just a single inequality, but a parameterized family of lower bounds.

→ Jensen's inequality can be used to define a variational inequality which swaps logarithm and summation.

This is relevant for mixture models as the per pattern log-likelihood is exactly of this form. Note that we get a separate variational inequality per data point, so we can chose a vector of convex weights $\mathbf{q}_t$ for each pattern $\mathbf{x}_t$ to get the so-called evidence lower bound (ELBO)

$$\ell(\boldsymbol{\theta}) = \sum_{t=1}^{s} \ln \sum_{z=1}^{k} \pi_z \, p(\mathbf{x}; \boldsymbol{\theta}_z) \geq \sum_{t=1}^{s} \sum_{z=1}^{k} q_{tz} \left[\ln \pi_z + \ln p(\mathbf{x}_t; \boldsymbol{\theta}_z) - \ln q_{tz}\right] \tag{ELBO}$$

We can now use the ELBO as an objective to be maximized: Maximizing with regard to the variational parameters will increase the tightness of the bound, whereas maximizing the ELBO with regard to the model parameters will improve the model fit.

→ The Evidence Lower BOund (ELBO) yields approximate, often tractable, objective functions for learning in latent variable models.

Let us solve for the optimal choice of the variational parameters. Note that the ELBO is separable with regard to the these parameters, i.e. we can solve every $\mathbf{q}_t$ independently. This means, we can drop the t-index and optimize the generic problem with $p_z = p(\mathbf{x};\boldsymbol{\theta}_z)$:

$$\mathbf{q} \overset{\max}{\to} \sum_{z=1}^{k} q_z \left[\ln p_z + \ln \pi_z - \ln q_z\right] - \lambda \left(\sum_z q_z - 1\right) \tag{3.8}$$

where we have introduced a Lagrange multiplier λ to enforce the normalization constraint on $\mathbf{q}$. The resulting first order optimality condition is

$$\ln p_z + \ln \pi_z - \ln q_z \overset{!}{=} \lambda + 1 \iff q_z \overset{!}{=} e^{-(\lambda+1)}\, \pi_z\, p_z \iff q_z \overset{!}{=} \frac{\pi_z\, p_z}{\sum_\zeta \pi_\zeta\, p_\zeta}\,. \tag{3.9}$$

The second equation is obtained through exponentiation and the third equation follows from the choice of the proportionality constant such that $\sum_z q_z = 1$. The solution has a very intuitive interpretation: it is the posterior of the latent class variable and the determining equation an instantiation of Bayes' rule. Note that the optimal choice of the variational parameters depends on the model parameters $\boldsymbol{\theta}$. It is only a partial solution step, called the Expectation (or E) step within an alternating maximization scheme.

→ The optimal choice of the variational distribution is the posterior of the latent variable (for fixed model parameters).

What can (structurally) be said about the maximization with regard to the model parameters $\boldsymbol{\theta}$? We can generally solve for $\boldsymbol{\pi}$ to get (after introducing a Lagrange multiplier and some algebra)

$$\pi_z = \frac{1}{s} \sum_{t=1}^{s} q_{tz} \tag{3.10}$$

The solution for $\boldsymbol{\theta}_z$ depends on the choice of model distribution, but we can generically get to separable problems

$$\boldsymbol{\theta}_z \overset{\max}{\to} \sum_{t=1}^{s} q_{tz} \ln p(\mathbf{x}_t;\boldsymbol{\theta}_z) \tag{3.11}$$

This means that parameters for different components z are decoupled given $\mathbf{q}$. For each component we have to simply solve a weighted MLE problem. It is often possible to analytically solve this problem, in which case we call this step a *maximization* (or M) step.[1]

→ The optimal choice of model parameters with regard to the ELBO reduces to finding weighted MLEs for each component.

We have derived the general framework of EM for mixture models. This can be generalized to other types of variables and models, which we will discuss subsequently. To conclude this section, we will consider a special case known as a *Gaussian mixture model.* Due to its importance, we include here an intermezzo on the normal distribution, which can be skipped without harm by the knowledgeable reader.

[1]More generally, one may have to resort to local improvements, which is called an incomplete M-step.

3.1.4 Normality

We provide here a basic introduction to the normal distribution as normal (aka Gaussian) random variables are pervasive in machine learning.

Binomial Distribution

What is the simplest source of randomness? It is a coin flip, a random bit. Let us perform a sequence of independent coin flips $X_1, X_2, \dots,$ with outcomes $x_1, x_2, \dots \in \{0, 1\}$ encoding 'tail' vs. 'head'. Let us now consider the partial sums,

$$S_n = \sum_{i=1}^{n} X_i, \quad 0 \le S_n \le n \tag{3.12}$$

which is itself a sequence of random variables. What is the distribution of S_n? Or equivalently: what is the probability to get k heads in a trial of length n? The answer is elementary: it is the binomial distribution

$$\mathbb{P}\{S_n = k\} = \binom{n}{k} p^k q^{n-k}, \quad q := 1 - p\,. \tag{3.13}$$

Here $p = \mathbb{P}\{X_i = 1\} \in [0; 1]$ is the success probability, e.g. $p = q = \frac{1}{2}$ for an unbiased coin. Let us shift and scale the partial sums, so they have zero mean and unit variance. It is easy to verify that

$$\mathbf{E}[S_n] = np, \quad \mathbf{E}[(S_n - np)^2] = npq, \quad \text{s.t.} \quad Z_n = \frac{S_n - np}{\sqrt{npq}}\,. \tag{3.14}$$

The variable sequence Z_n is centered and *standardized*. As is well-known for large enough n and large enough pq (p not being too close to the extremes) the binomial distribution (with its unwieldy factorials) can be numerically approximated by a normal distribution with zero mean and unit variance. In general:

$$X \sim \mathcal{N}(\mu, \sigma^2), \quad \text{with density} \quad p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]. \tag{3.15}$$

This can often be illustrated by plotting the binomial law together with the Gaussian bell shape of the normal density (cf. Figure 3.1).

Central Limit Theorem

However, the above observation is more than just a convenient computational trick for a special case of distributions. The deeper reason is the *central limit theorem* (CLT), which holds for *all* standardized partial sums of iid random variables. This is a strong universality property.

$$Z_n \overset{d}{\to} \mathcal{N}(0, 1), \qquad \overset{d}{\to} \equiv \text{ converges in distribution to} \tag{3.16}$$

In the one-dimensional case, we can think of this as pointwise convergence of the cumulative distribution functions (at continuity points).

→ The standardized partial sums of a sequence of iid random variables converges in distribution to a normal distribution (Central Limit Theorem).

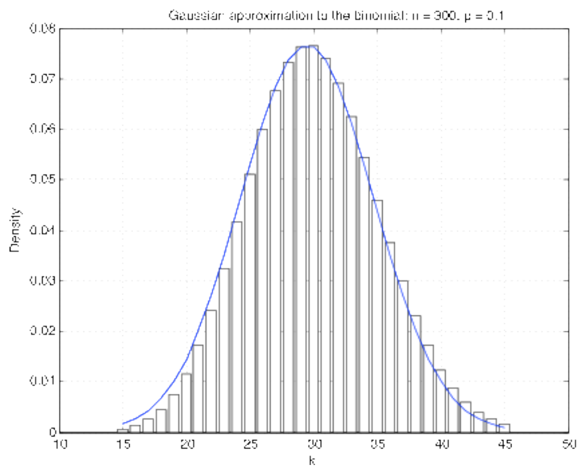


Figure 3.1: Gaussian approximation to the binomial.

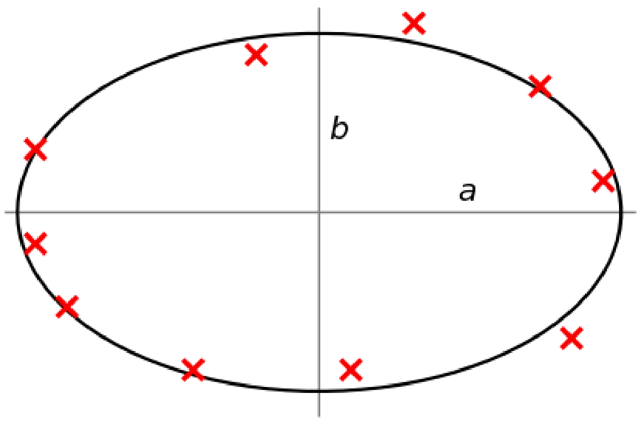


Figure 3.2: Fitting an ellipse to a set of points with least squares. Cf. `https://en.wikipedia.org/wiki/Least_squares`

The normal distribution is special! Intuitively speaking, whenever we are 'washing out' the randomness by averaging, the shape of the distribution of the elementary random events becomes irrelevant. The CLT also holds in many cases where distributions are not necessarily identical but fulfill certain conditions (e.g. the Lindeberg condition). Another very important generalization of the CLT is to martingales. These are sequences (processes) where Z_{n+1} may depend on $Z_1, \ldots, Z_n$, but it is required that $\mathbf{E}[Z_{n+1} - Z_n | Z_1, \ldots, Z_n) = 0$.

R This CLT can be extended to non-iid data, for instance by considering martingales. This can be useful in a lot of asymptotic analyses as establishing the martingale property may be relatively straightforward.

Least Squares

A second feature of the normal distribution is its connection to least squares. The following regression technique goes back to Gauss: assume that (i) we have a curve f_θ parameterized by θ (e.g. a line or quadratic) and (ii) a set of points $\{\mathbf{x}_i\}$, then a least squares fit is given

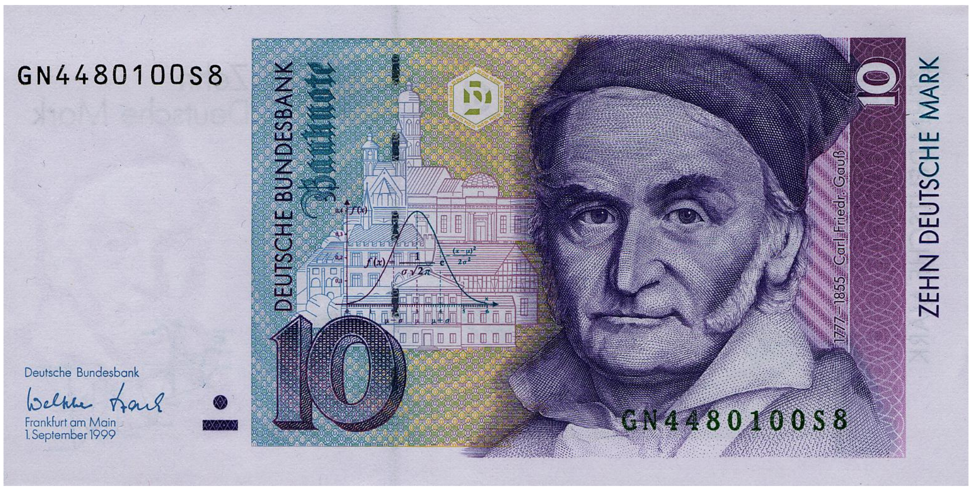


Figure 3.3: A 10 Deutsche Mark bill showing Gauss and the normal density (Gaussian bell curve).

by

$$\theta \xrightarrow{\min} \frac{1}{2n} \sum_{i=1}^{n} \min_{\mathbf{x} \in f_\theta} (\mathbf{x} - \mathbf{x}_i)^2 . \tag{3.17}$$

An example is shown in Figure 3.2. In the special case, where f_θ is a function and $\mathbf{x} = (x, y)$

$$\theta \xrightarrow{\min} \frac{1}{2n} \sum_{i=1}^{n} (f_\theta(x_i) - y_i)^2 \tag{3.18}$$

What does this have to do with the normal distribution? Note that if we assume that data points are corrupted by normal (or Gaussian – sic!) noise, then the least squares solution is the one that maximizes the probability (probability density function value) of the observed data. Hence the use of the squared error metric and assumptions about the normality of noise go hand in hand. What used to go *from* hand to hand is the bill shown in Figure 3.3. Gaussian densities used to be common knowledge and a bank note could serve as a cheat sheet :)

→ Normal (noise) distributions are often implicit in the use of least squares criteria for estimation.

Multivariate Normal

In probabilistic modeling, the multivariate Gaussian is omnipresent.

Q How we can derive a multivariate version of the normal distribution?

Let us show how we can lift from one to m dimensions. The idea is to consider a vector of iid Gaussian random variables $x_j \sim \mathcal{N}(\mu_j, \sigma_j^2)$ and to calculate

$$p(\mathbf{x}; \{\mu_j, \sigma_j^2\}) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right] \tag{3.19}$$

$$= \frac{\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]}{(2\pi)^{\frac{m}{2}} \det(\boldsymbol{\Sigma})^{\frac{1}{2}}}, \quad \boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2) \tag{3.20}$$

This shows how to switch to a vector/matrix view by lifting the product into the exponent. The formula derived generalizes to choices of $\boldsymbol{\Sigma}$, aka the variance-covariance matrix, as any positive semi-definite matrix. The entries of $\boldsymbol{\Sigma}$ have a very precise interpretation

$$\sigma_{ij} = \mathbf{E}[x_i x_j], \quad \sigma_{ij} = 0 \iff x_i \text{ and } x_j \text{ are (marginally) independent and} \tag{3.21}$$

$$(\boldsymbol{\Sigma}^{-1})_{ij} = 0 \iff x_i \text{ and } x_j \text{ are conditionally independent on } \{x_k : k \neq i, j\}$$

The inverse of $\boldsymbol{\Sigma}$ is also called the precision matrix. The contour lines of the density function are hyper-ellipsoids. In the diagonal (and general) case they are given as the solution to the quadratic equation (after centering $\mathbf{x} \leftarrow \mathbf{x} - \boldsymbol{\mu}$)

$$\sum_j \left(\frac{x_j}{\sigma_j} \right)^2 = \text{const} \quad \text{more generally} \quad (\mathbf{U}\mathbf{x})^\top (\mathbf{U}\mathbf{x}) = \text{const}, \quad \boldsymbol{\Sigma}^{-1} = \mathbf{U}^\top \mathbf{U}\,. \tag{3.22}$$

3.1.5 Gaussian Mixture Model

Let us specialize the mixture model to the Gaussian case. One typically uses Gaussian component models of fixed (for simplicity unit) variance, leading to

$$p(\mathbf{x}; \boldsymbol{\pi}, \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}) = \sum_{z=1}^k \pi_z p(\mathbf{x}; \boldsymbol{\mu}_z), \quad p(\mathbf{x}; \boldsymbol{\mu}_z) = (2\pi)^{-\frac{d}{2}} \exp\left[-\frac{1}{2} \|\mathbf{x} - \boldsymbol{\mu}_z\|^2 \right]. \tag{3.23}$$

A picture showing a simple $k = 5$ mixture with generated samples is shown in Figure 3.4. The EM algorithm then consists of the following alternating equations

$$\mathbb{P}(Z_t = z | \mathbf{x}_t, \boldsymbol{\theta}) = \frac{\pi_z \exp\left[-\frac{1}{2}\|\mathbf{x}_t - \boldsymbol{\mu}_z\|^2\right]}{\sum_\zeta \pi_\zeta \exp\left[-\frac{1}{2}\|\mathbf{x}_t - \boldsymbol{\mu}_\zeta\|^2\right]} =: q_{tz} \tag{E-step}$$

$$\boldsymbol{\mu}_z = \frac{\sum_{t=1}^s q_{tz} \mathbf{x}_t}{\sum_{t=1}^s q_{tz}}, \quad \pi_z = \frac{1}{s} \sum_{t=1}^s q_{tz} \tag{M-step}$$

As said above, we can interpret the E-step posteriors as soft clustering weights. The Gaussian means are re-calculated in the M-step as the weighted centroids of the associated data points. It can be seen that the EM algorithm for Gaussian mixtures is closely related to the k-means algorithm, which is recovered in the limits of $\sigma \to \infty$.

→ The EM algorithm for the Gaussian mixture model alternates between updating probabilistic cluster membership and re-computing cluster centers with a weighted centroid condition.

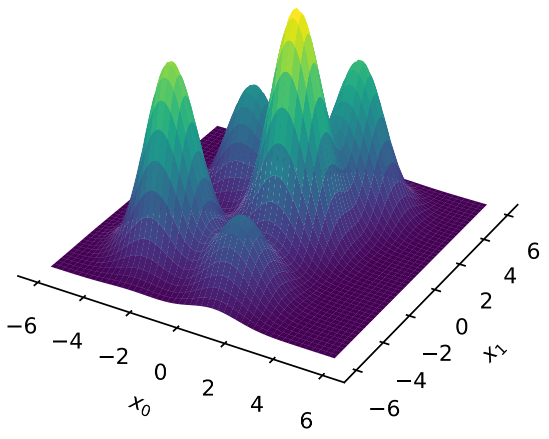


Figure 3.4: Example of a Gaussian $k = 5$ mixture.

3.2 Topic Models

We now turn to a related, but different class of latent variable models known as *topic models*. They take their name from the paradigmatic use case in which they have been developed: to analyse document collections and to discover the *topical content* of a document, its *aboutness*.

→ *Topical content* is the information words carry on "what a text is about". It is assumed to be invariant with respect to word order.

As such, topic models aim to explicitly discover linguistic semantics, however, they have also been used to discover, for instance, interest patterns in collaborative filtering. We pursue a step-by-step approach here driven by the question:

Q How can we formalize topical semantics as a statistical model?

The educational point of this exercise is to make visible the interaction between intuitions about the problem (*what are topics?*), its mathematical formalization (*what are topic models?*), and its statistical or computational realization (*how can we learn topic models?*).

3.2.1 Order Invariance and Exchangeability

We will treat a document collection as a field of random variables X_{it} with realizations $x_{it} \in \Sigma$, where Σ, $|\Sigma| = m$ is the word vocabulary, $i = 1, \dots, n$ enumerates documents and $t = 1, \dots s_i$ word positions in the i-th document. So in the example below, say a document contains $s_i = 13$ words, then we think of each slot as a random variable and the observed word from the vocabulary Σ as its realization.

$$\underbrace{\text{When}}_{X_{i1}}\ \underbrace{\text{the}}_{X_{i2}}\ \underbrace{\text{blackberries}}_{X_{i3}}\ \underbrace{\text{hang}}_{X_{i4}}\ \underbrace{\text{swollen}}_{X_{i5}}\ \underbrace{\text{in}}_{X_{i6}}\ \underbrace{\text{the}}_{X_{i7}}\ \underbrace{\text{woods}}_{X_{i8}}\ \underbrace{\text{in}}_{X_{i9}}\ \underbrace{\text{the}}_{X_{i10}}\ \underbrace{\text{brambles}}_{X_{i11}}\ \underbrace{\text{nobody}}_{X_{i12}}\ \underbrace{\text{owns}}_{X_{i13}}$$

The first thing to consider is:

Should word order matter in topic models?

It is a defining feature of topic models – as opposed to other semantic models – to ignore word order and to focus only on the usage of words at a document or paragraph level. We thus assume: the aboutness of a document is described by the words it uses, irrespective of its propositional content. This assumption is debatable in its radicality, but leads to important modeling simplification. Mathematically speaking, such an invariance to observation order is known as *exchangeability*, which says that the distribution of a sequence of random variables $X_1, X_2, \ldots, X_s$ does not change under any permutation of their order.

→ Topic models ignore word order and make an exchangeability assumption.

Are there relevant consequences of this assumption? It is clear that any inference about the distribution – something we want to learn or estimate – should not depend on non-invariant aspects of the data. So let us first identify those statistics, i.e. functions of the data, that are invariant, the so-called *sufficient statistics*.

What are the sufficient statistics of topic models?

It is intuitively clear that these statistics are the frequencies of occurrence of words in documents. Retaining more than just the number of occurrences will have to refer to positional or order information and retaining less, e.g. only which terms occurred and which ones not, will possibly lead to a loss of statistical efficiency. We can hence reduce the data from the start to what is known as a bag-of-words representation of occurrence counts, formally

$$N_{ij} = \left|\{x_{it} = w_j \mid t = 1, \ldots, s_i\}\right| . \tag{3.24}$$

So N_{ij} denotes how often word w_j occurred in document d_i.

→ We can summarize a document corpus in an occurrence matrix of counts

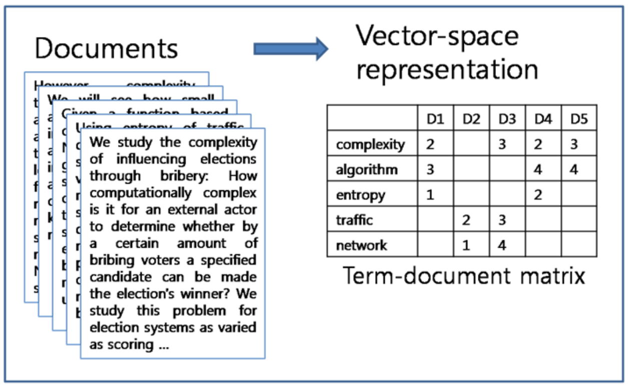
$$\mathbf{N} = (N_{ij}) \in \mathbb{Z}_{\geq 0}^{n \times m} . \qquad \text{(occurrence matrix)}$$

We have also depicted this reduction pictorially in Figure 3.5.

There is another, deeper consequence of exchangability, which is known as de Finetti's theorem, which will come back to later. This theorem shows that the constraints imposed on dependencies by the exchangablity property leads to a particular representation as continuous mixtures

$$p(X_1, \ldots, X_s) = \int \prod_{r=1}^{s} p(x_r; \boldsymbol{\theta})\, dP(\boldsymbol{\theta}) \tag{3.25}$$

This means that exchangeable distributions always have a representation involving a latent parameter $\boldsymbol{\theta}$, conditioned on which observations are rendered independent. While this may seem mysterious here, we will re-visit this in the context of the Latent Dirichlet Allocation model.



We study the complexity of influencing elections through bribery: How computationally complex is it for an external actor to determine whether by a certain amount of bribing voters a specified candidate can be made the election's winner? We study this problem for election systems as varied as scoring ...

	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Figure 3.5: Occurrence matrix, also known as term-document matrix in information retrieval.

3.2.2 Probabilistic Latent Semantics Model

Q What are topics? What is the complete data model?

We have to conceptualize how the topical content should be represented formally. What do we mean by a topic? In the simplest case, we can say that different topics are discrete units such as "yoga", "travel", or "photography", things that people are interested in to talk or write about.

→ Topics are nominal categories of aboutness.

So formally, we can think of topics as being modeled by a variable $z \in \{1, \ldots, k\}$, similarly to the case of mixture models.

R This of course assumes that the number of topics is finite and known, that there is no structure – say hierarchical, etc. In a practical context, one may want to consider revisiting some of these issues. Note that we want to attach topical content to each word *occurrence* and not each word *per se*.

We now need to decide on:

Q What to attach topic variables to?

If we were to attach it to an entire document, we would effectively perform document clustering. A somewhat more interesting model, the one followed in topic models, is to introduce a topic variable for each word occurrence. Let us reflect on that: this means that the usage of a word in the context of a document makes reference to a topic. This allows for the fact that the same word can refer to different topics in different contexts. This is a feature as word can be ambiguous (homography, polysemy, references). Examples: "bear" (animal vs. verb), "wood" (material vs. area), "Michael Jordan" (basketball player vs. machine learning researcher). It also means that the topics of documents will be induced by the topics of the words they contain. This should also be considered a feature as topics need not to be mutually exclusive, but can be combined. For instance, a document on *soccer world cup 2022 in Dubai* may contain soccer vocabulary (e.g. "teams", "play", "soccer", "match"), but also political vocabulary (e.g. "labor", "corruption", "president"), or medical terms (e.g. "pandemic", "Covid")

Topic variables are associated with each word occurrence.

So formally the complete data will be a sequence of word-topic pairs for each document:

$$(X_{it}, Z_{it}), \quad X_{it} \in \Sigma, \; Z_{it} \in \{1, \ldots, k\} \tag{3.26}$$

The next question is how to define a model? In the spirit of latent variable models, this involves to parts: (1) Define a distribution over the latent (i.e. topic) variables. (2) Define a conditional distribution of a word given a topic. Let us consider the latter first. A saturated model would be one that simply enumerates the probability of each word under each topic. This seems plausible as topics would then be characterized by the words that are "about" them.

Topics are represented as distributions over words, i.e. conditionals $p(w|z)$.

This leaves the question of how to define the distribution over the latent variables. It is clear, we cannot condition on the word occurrence as this is generated downstream. The only information one can condition on is hence the document identity.

Each document is characterized by a distribution over topics $p(z|d)$

Putting these pieces together, we can formulate the topic model as a two stage sampling process, where for each occurrence, we (1) sample a topic from $p(z|d)$, and (2) sample a token, given the sampled topic $p(w|z)$. This is shown in Figure 3.6. we can also write this

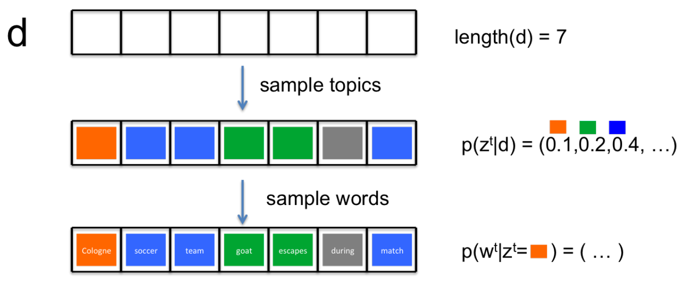


Figure 3.6: Probabilistic Latent Semantic Analysis: two-stage sampling

basic topic model, also called probabilistic Latent Semantic Analysis (pLSA), in a formula as a probabilistic model over occurrence matrices. We get the log-likelihood

$$\ell(\boldsymbol{\theta}; \mathbf{N}) = \ln p(\mathbf{N}; \boldsymbol{\theta}) = \sum_{i,j} N_{ij} \log p(w_j|d_i), \quad p(w_j|d_i) = \sum_{z=1}^{k} p(w_j|z) p(z|d_i) \tag{3.27}$$

“segment 1”	“segment 2”	“matrix 1”	“matrix 2”	“line 1”	“line 2”	“power 1”	“power 2”
imag	speaker	robust	manufactur	constraint	alpha	**POWER**	load
SEGMENT	speech	**MATRIX**	cell	**LINE**	redshift	spectrum	memori
texture	recogni	eigenvalu	part	match	**LINE**	omega	vlsi
color	signal	uncertainti	**MATRIX**	locat	galaxi	mpc	**POWER**
tissue	train	plane	cellular	imag	quasar	hsup	systolic
brain	hmm	linear	famili	geometr	absorp	larg	input
slice	source	condition	design	impos	high	redshift	complex
cluster	speakerind.	perturb	machinepart	segment	ssup	galaxi	arrai
mri	**SEGMENT**	root	format	fundament	densiti	standard	present
volume	sound	suffici	group	recogn	veloc	model	implement

Table 3.1: Eight selected topics from a 128 topic decomposition. The displayed word stems are the 10 most probable words in the class-conditional distribution $p(\text{word}|\text{topic})$, from top to bottom in descending order.

3.2.3 Expectation Maximization Algorithm

We will now derive the EM algorithm for the topic model introduced above. Note first that we can equivalently write the log-likelihood based on the raw data

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{t=1}^{s_i} \ln p(x_{it};\boldsymbol{\theta}), \quad p(x_{it};\boldsymbol{\theta}) = \sum_{z=1}^{k} p(x_{it}|z)p(z|d_i) \tag{3.28}$$

We can now apply the standard variational bound to each of the log-probabilities. This results in the ELBO

$$\ell(\boldsymbol{\theta}) \geq \sum_{i=1}^{n} \sum_{t=1}^{s_i} \sum_{z=1}^{k} q_{itz} \left[\ln p(x_{it}|z) + \ln p(z|d_i) - \ln q_{itz}\right] =: \hat{\ell}(\boldsymbol{\theta};\mathbf{q}) \tag{3.29}$$

The structure of the ELBO is very similar to case of mixture models. Following essentially the same steps, but taking the specific normalization constraints into account, we can derive the EM equations

$$q_{itz} = \frac{p(x_{it}|z)p(z|d_i)}{\sum_{\zeta} p(x_{it}|\zeta)p(\zeta|d_i)} \tag{E-step}$$

$$p(w_j|z) = \frac{\sum_{i,t} \mathbb{I}[x_{it} = w_j]\; q_{itz}}{\sum_{i,t} q_{itz}}, \qquad p(z|d_i) = \frac{1}{s_i}\sum_{t=1}^{s_i} q_{itz} \tag{M-step}$$

The EM equations for the topic model can be derived in close analogy to mixture models.

Interpreting these equations, we see that the E-step gives posterior probabilities, where $p(z|d_i)$ acts as a prior. This yields a probabilistic clustering of word occurrences (sic! not of words or documents). The M-step equations are natural as they are just MLEs for a $\mathbf{q}$-weighted multinomial sample. Note that all updates are sparse, scaling with the total number of word occurrences $s = \sum_i s_i$. The EM algorithm will converge, but there is no guarantee that the solution will be a global maximizer of the log-likelihood.

Finally, we provide some anecdotal evidence that the topic model is identifying interesting regularities in document collections in Table 3.2.3. This is based on a small corpus of research articles, words have been stemmed.

3.2.4 Latent Dirichlet Allocation

The topic model presented above assumes that a fixed set of documents is given and then selects the parameters so as to maximize the predictability of words within a document, which is what the likelihood criterion effectively represents. One may wonder:

Q How can the topic model be extended in a way that accounts for modeling new, unseen documents?

An answer to this question is given by the so-called Latent Dirichlet Allocation (LDA) model [3]. Let us introduce a proper parameter notation for the conditional distributions in the topic model

$$u_{jz} := p(w_j|z), \quad v_{zi} := p(z|d_i) \tag{3.30}$$

The main idea is to define a distribution over per-document topic mixture vectors $\mathbf{v} \in [0;1]^k$, $\sum_z v_z = 1$. What distribution is appropriate? There are many choices, but one way to get to a unique answer is via the principle of *conjugacy*. In Bayesian inference a *conjugate prior* is one that is compatible with the likelihood in a way that it leads to a posterior belonging to the same family. This is also known as the reproducing property. The conjugate choice to the categorical topic variable turns out to be a Dirichlet distribution, which has density proportional to

$$p(\mathbf{v}|\boldsymbol{\alpha}) \propto \prod_{z=1}^{k} v_z^{\alpha_z - 1}, \quad \alpha_z > 0 \tag{3.31}$$

In the simplest case, we will assume that the hyperparameters are chosen $\alpha_z = \alpha$, where α can be optimized on a held-out data set.

→ The Dirichlet distribution is a prior distribution over topic mixture weights that can be motivated from the principle of conjugacy.

We now need to augment the topic model with a Dirichlet prior to arrive at the model known as Latent Dirichlet allocation (LDA). The basic idea is to integrate over the document-specific parameters – or technically speaking, to treat them as nuisance parameters – and to only retain the topic distributions over words, i.e. the parameter matrix $\mathbf{U}$. Executing this plan, we get a model for a fixed length[2] document

$$\mathbb{P}(X = (x_1, ..., x_s); \mathbf{U}) = \int \prod_{t=1}^{s} p(x_t|\mathbf{U}, \mathbf{v})\, p(\mathbf{v}|\boldsymbol{\alpha}) d\mathbf{v}, \;\; p(w_j|\mathbf{U}, \mathbf{v}) = \sum_{z=1}^{k} u_{jz} v_z \tag{3.32}$$

Here we see that we retain the same word prediction model, but instead of providing an MLE point estimate, we perform Bayesian averaging to get the Bayesian predictive distribution.

→ The Latent Dirichlet Allocation model uses a Bayesian predictive distribution to model topics.

[2]One can further extend the model by making s a random variable.

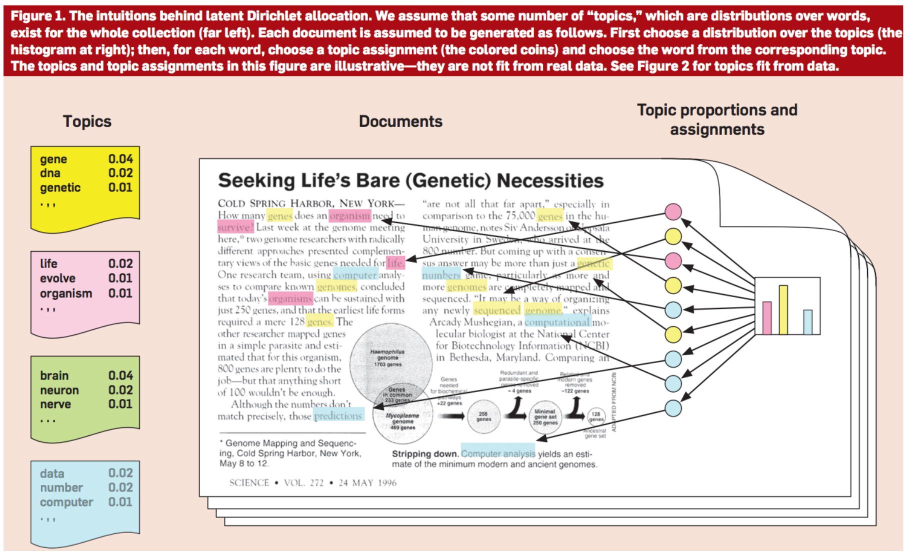


Figure 1. The intuitions behind latent Dirichlet allocation. We assume that some number of "topics," which are distributions over words, exist for the whole collection (far left). Each document is assumed to be generated as follows. First choose a distribution over the topics (the histogram at right); then, for each word, choose a topic assignment (the colored coins) and choose the word from the corresponding topic. The topics and topic assignments in this figure are illustrative—they are not fit from real data. See Figure 2 for topics fit from data.

Figure 3.7: Intuition behond LDA, Figure taken from [2].

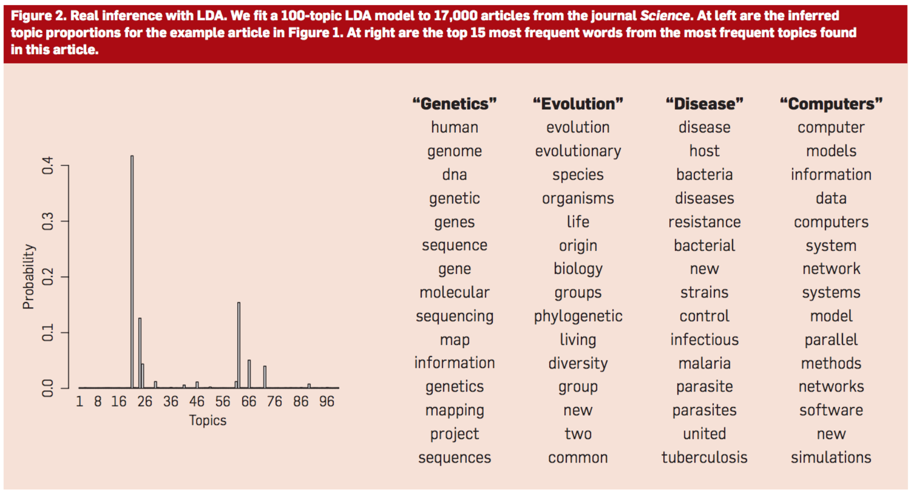


Figure 2. Real inference with LDA. We fit a 100-topic LDA model to 17,000 articles from the journal *Science*. At left are the inferred topic proportions for the example article in Figure 1. At right are the top 15 most frequent words from the most frequent topics found in this article.

"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

Figure 3.8: Topics inferred by LDa. Figure taken from [2].

LDA offers a somewhat more robust and preferred topic model. It is somewhat more involved to develop scalable learning algorithms for LDA, but there are nowadays many highly scalable implementations available that rely on Markov Chain Monte Carlo (collapsed Gibbs sampling) or variational EM. Finally, we reproduce a exemplary results from [2] in Figure 3.7 and 3.8.

3.2.5 Probabilistic Matrix Decomposition

Q How does the probabilistic latent semantics model relate to matrix decomposition?

When formulating a model one is often guided and motivated by particular use case (as above for topic models). However, it is important – and in itself not always obvious how – to relate a model to other, simpler or pre-existing models. In the case of the topic model above, it seems that the topic variable plays the role of a dimension (or rank) bottleneck.

Can we make this connection more precise? The connection becomes more obvious, in the explicit parameterization:

$$u_{jz} := p(w_j|z), \quad v_{zi} := p(z|d_i)\,. \tag{3.33}$$

We can then form $\hat{\mathbf{N}} = \mathbf{UV}$, which is a rank k (or less) matrix. This matrix is not a matrix of counts, but we can think of it as an approximation of relative word frequencies, more precisely $N_{ij} \approx s_i \hat{N}_{ij}$.

→ The probabilistic latent semantics model constructs a low rank approximation to the observed count matrix.

If we take the probabilistic semantics into account, we have to consider additional constraints in the parameter space. First of all, parameters are non-negative

$$u_{jz} \geq 0 \quad (\forall j, z), \qquad v_{zi} \geq 0 \quad (\forall i, z) \tag{3.34}$$

A factorization of a matrix into non-negative factors is known as *non-negative matrix factorization* (NMF). Note that the objective directly follows from the latent variable modeling approach and is not imposed in an *ad hoc* manner. In the matrix view we can re-write the log-likelihood objective as

$$\ell(\mathbf{U}, \mathbf{V}; \mathbf{N}) = \sum_{i,j} N_{ij} \ln \hat{N}_{ij}, \quad \hat{\mathbf{N}} = \mathbf{UV} \tag{3.35}$$

In addition to the non-negativity constraints, there are also further normalization constraints on the row/column level, namely

$$\sum_j p(w_j|z) = \sum_j u_{jz} = 1, \quad (\forall z) \qquad \sum_z p(z|d_i) = \sum_z v_{zi} = 1, \quad (\forall i)\,. \tag{3.36}$$

Again, these additional constraints may appear somewhat arbitrary in the context of (non-negative) matrix factorization, but are a direct consequence of the underlying statistical model and its probabilistic interpretation.

→ The topic model can be thought of as a special case of a non-negative matrix decomposition, but with a principled log-likelihood objective.

3.2.6 Non-Negative Matrix Factorization

Let us finally consider non-negative matrix factorization in its own right and independent of topic models and their motivation. In many cases we observe matrices that are non-negative, for instance count matrices like above or just real-valued matrices with non-negative entries, e.g. measurements like intensities that by definition cannot be negative. We would then like to make sure the approximation or reconstruction of a matrix is guaranteed to be non-negative. Constraining the matrix factors to be non-negative is one avenue to build this directly into the model.

Let us consider the problem of approximating a matrix $\mathbf{A}$ with positive factors, using the squared objective as a criterion

$$\ell(\mathbf{U}, \mathbf{V}) = \frac{1}{2}\|\mathbf{A} - \mathbf{UV}\|_F^2, \quad \mathbf{U}, \mathbf{V} \geq 0\,, \tag{3.37}$$

where for simplicity we focus on the fully observed case. We ask:

How is a non-negative matrix decomposition different from an unconstrained one?

The non-negativity constraints may superficially look relatively minor, however, they completely change the nature of the matrix decomposition as they prevent cancellations of entries in sums. If we think of images, then each factor produces (loosely speaking) "ink" for an image, which cannot be erased by other factors. The superposition of factors adds up with the same (positive) sign. This means that the inferred decompositions are typically part-based in nature [30]. To illustrate this point, we show factors learned by NMF, clustering (vector quantization) and PCA in Figure 3.9. It is clear that clustering produces prototypical faces in their entirety, while NMF produces factors that correspond to different parts of the face, which then get combined in a relatively sparse manner. The PCA decomposition makes use of cancellation effects (red color). A higher resolution example is shown in Figure 3.10. This is a very educating example to see how small adjustments in the problem formulation can have very significant effects on the solutions that are found.

→ Non-negative matrix factorization often identifies factors that are qualitatively different from PCA in that they tend to identify part-based representations.

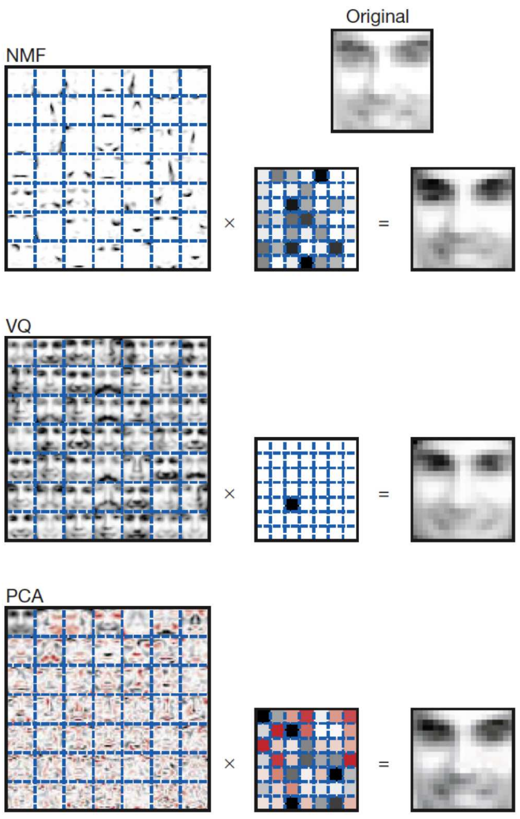


Figure 3.9: Factors identified by NMF, VQ, and PCA. Taken from [30].

Let us finally quickly touch on computational aspects. One algorithmic option is to use the Alternating Least Squares algorithm. It basically only needs to be augmented by

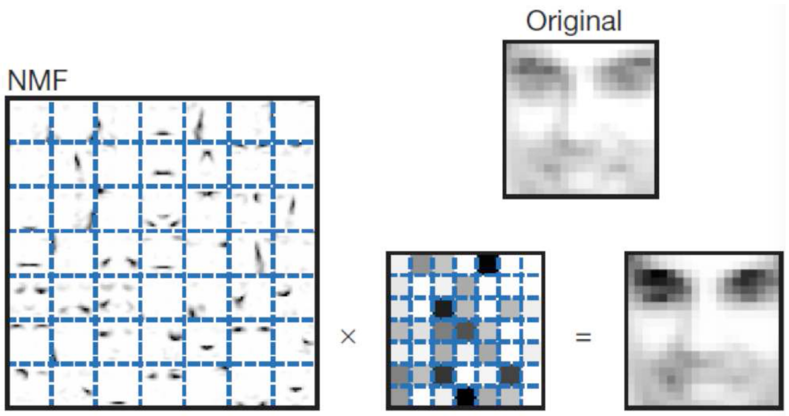


Figure 3.10: Larger resolution of how to approximate a face by an NMF factorization.

projection steps that enforce non-negativity. This can be done through clipping.

$$u_{jz} = \max\{0, u_{jz}\}, \quad v_{zi} = \max\{0, v_{zi}\} \tag{3.38}$$

For a more detailed discussion of algorithms for NMF see [1].

3.3 Embeddings

3.3.1 Motivation

One of the fundamental problems in learning with symbolic or discrete data is to map atoms (i.e. discrete units) to vector space representation also known as *embeddings*. As a running example we will refer to the problem of lexical semantics, namely how to learn semantic representation for words (or multi-word phrases). This is an interesting problem as: (1) The link between the word (as a sequence of phonems or graphems) and its meaning is *conventional*. As such it has to be learned either from a dictionary definition/gloss and/or from the usage of words in language. The latter has been a topic in the philosophy of language, for instance in the writings of Ludwig Wittgenstein, who states that "in most cases, the meaning of a word is its use". This open up interesting avenues for machine learning. (2) The meaning of a word is not explicitly represented in language. Meaning is not tied to a word in a one-to-one fashion, not even in a many-to-one or many-to-many manner, but it depends on the linguistic context and even on non-linguistic aspects of the situation in which language is used as a means of communication. This leaves open the question of how to even represent meaning.

3.3.2 Word2Vec

Let us first consider the problem of learning semantic word representations in a context-independent manner. We would like to learn a mapping from words to vectors

$$\underbrace{\Sigma \ni \mathbf{w}}_{\text{word}} \mapsto \underbrace{\mathbf{z}_{\mathbf{w}} \in \mathbb{R}^m}_{\text{vector}} \tag{3.39}$$

where m is some chosen dimensionality, typically in the range of $m = 100 - 500$ in practice.

Q How can we learn a mapping from words to vectors such that vectors represent word semantics?

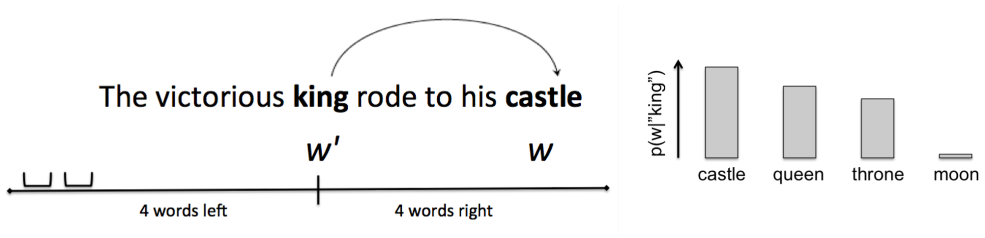


Figure 3.11: Illustration of the skip-gram model

The first problem with this question – similar to what we have seen for topic models – is to formally define what is meant by "semantics". There is no generally accepted semantic formalism, which a machine would simply have to re-discover. One possible solution is to make use of the latent variable approach and to think of the latent representation of being predictive of what is observed. We will see that we can use a very un-informative prior on the latent representations (in fact ignore the prior altogether), but we need to answer the crucial question of what it is that we want to predict. How can we quantify, whether a representation is capturing semantics? One idea is to follow the verdict of Firth: "You shall know a word by the company it keeps". Here we will interpret "company" as the context of a word, in the simplest form the other words that co-occur with the word in question within a fixed-size window.

→ Treat the embedding of each word as a latent variable that predict co-occurring context words.

We illustrate this in Figure 3.11. We aim to learn a probabilistic model – also known as *distributional semantics* – of co-occurring words. This model is also called the *skip-gram model*.[3] Let us formally write down the likelihood function for a sequence of words $\mathbf{x} = x_1, \ldots, x_T$, making an iid. assumption and encoding the window in a set of positional displacements $\mathcal{I}$:

$$\ell(\theta; \mathbf{x}) = \sum_{t=1}^{T} \sum_{\triangle \in \mathcal{I}} \log p(x_{t+\triangle} \,|\, x_t; \boldsymbol{\theta}) \tag{3.40}$$

We will also assume suitable padding and identify x_t with a special token for $t < 1$ and $t > T$. Note that – for conreteness – a symmetric window of size $2R$ corresponds to the choice $\mathcal{I} = \{-R, \ldots, -1, 1, \ldots R\}$.

Q How should we relate the prediction of skip-grams to latent word embeddings?

We now come to the key aspect of the word2vec model. We want to express $p(v|w)$ for word pairs v, w for which we have embedding vectors $\mathbf{z}_v$ and $\mathbf{z}_w$. What is the simplest scalar function that we can compute from two vectors? Arguably it is the inner product $\langle \mathbf{z}_w, \mathbf{z}_v \rangle$. Note that the latter is a *bi-linear* function to the reals. Generally we prefer probabilistic models to be linear on the log-scale and hence it is natural to presuppose

$$\ln p(v \,|\, w) = \langle \mathbf{z}_w, \mathbf{z}_v \rangle + const. \quad \Longrightarrow \quad p(v \,|\, w) = \frac{\exp[\langle \mathbf{z}_w, \mathbf{z}_v \rangle]}{\sum_u \exp[\langle \mathbf{z}_w, \mathbf{z}_u \rangle]} \tag{3.41}$$

[3]This is in reference to n-gram models that capture statistics of n consecutively occurring words. Here we are dealing with pairs that can be separated by 'skipped-over' words.

We consider a (log-)bi-linear model of the word embeddings.

One often introduces a parameterization that explicitly includes biases $b_v \in \mathbb{R}$. This allows to more explicitly control the marginal probability of the generated word. Moreover, nothing speaks against increasing the modeling power by using different embedding vectors for the conditioned word vs. the predicted word. While this is not strictly necessary in a minimalistic view, it has practical benefits. We thus arrive at the model

$$p(v \mid w; \boldsymbol{\theta}) = \frac{\exp[\boldsymbol{\zeta}_w^\top \mathbf{z}_v + b_v]}{\sum_{w'} \exp[\boldsymbol{\zeta}_w^\top \mathbf{z}_{w'} + b_{w'}]}, \quad w \mapsto \boldsymbol{\theta}_w = (\mathbf{z}_w, \boldsymbol{\zeta}_w, b_w) \in \mathbb{R}^{2m+1} \qquad \text{(word2vec)}$$

A refined model includes bias parameters and two embeddings per word.

Note that the model is invariant with respect to the absolute position of observation pairs as long as they co-occur within the defined context. One can thus define the sufficient statistics

$$N_{vw} := |\{t : \; x_t = w, \; x_{t+\triangle} = v, \; \triangle \in \mathcal{I}\}| \tag{3.42}$$

with which we can express the log-likelihood function as follows:

$$\ell(\boldsymbol{\theta}; \mathbf{x}) = \sum_{v,w} N_{vw} \Big(\langle \boldsymbol{\zeta}_w, \mathbf{z}_v \rangle + b_v \; - \ln \underbrace{\sum_{u \in \Sigma} \exp[\boldsymbol{\zeta}_w^\top \mathbf{z}_u + b_u]}_{\text{normalization constant}} \Big). \tag{3.43}$$

One can maximize this function with generic first order methods such as gradient descent. However, for large scale applications it is very inconvenient to have to compute the normalizing constants in every step as they scale with the size of the vocabulary $|\Sigma|$.

Are there alternate choice for the objective function that avoid the expensive computation of normalizing constants?

3.3.3 Negative Sampling

The original word2vec paper [31] proposes to overcome the complexity bottleneck by reducing the prediction problem to a classification problem. Note that if our goal is to learn representations, the actual prediction task we consider to learn the representations is not prescribed. There is *a priori* no right or wrong. We are hence free – modulo to practical/experimental success – to modify the prediction problem. For an observed pair of word (v, w) let us consider the binary classification model known as the *logistic* model

$$p(v \mid w) = \sigma(v, w; \boldsymbol{\theta}) := \sigma(\langle \boldsymbol{\zeta}_w, \mathbf{z}_v \rangle + b_v), \quad \sigma(z) = \frac{1}{1 + \exp[-z]} \in (0; 1). \tag{3.44}$$

We can interpret this as the probability for v to co-occur in a context with w. The probability will be larger for more frequent words v, which can be controlled explicitly by b_v, as well as for words that have a large inner product in their word vectors with the conditioning word w. It is then natural to define the positive multiset of examples as the actually co-occurring pairs

$$\mathcal{S}^+ = [(x_t, x_{t+\triangle}) : t = 1, \ldots, T, \; \triangle \in \mathcal{I}] \tag{3.45}$$

However, if we want to formulate a classification problem, we also need negative examples, otherwise the problem would degenerate.

Q In setting up a classification problem for skip-grams, how can we define negative examples?

One simple way to create negative examples is via sampling random pairs.

$$\mathcal{S}^- = \{(x_t, v_{tj} : t = 1, \dots, T,\ v_{tj} \overset{\text{iid}}{\sim} q,\ j = 1, \dots, r\} \tag{3.46}$$

This can be interpreted as follows: (1) One selects each word occurrence in the corpus with the same frequency r. (2) For each occurrence x_t one select r words (iid) at random as negative context words.[4] (3) All negative words are drawn from the same distribution q independent of context. Now we can use standard statistical inference to define an objective, namely the logistic log-likelihood

$$\ell(\boldsymbol{\theta}; \mathbf{x}) = \sum_{(w,v)\in\mathcal{S}^+} \ln \sigma(v, w; \boldsymbol{\theta}) + \sum_{(w,u)\in\mathcal{S}^-} \ln(1 - \sigma(u, w; \boldsymbol{\theta})) \tag{3.47}$$

$\rightarrow$ The classification problem for skip-grams can be formalized by taking the logistic log-likelihood as an objective function.

Obviously, if for positive examples, i.e.‘co-occurring word pairs, we want the probability of co-occurrence under our model to be high (close to 1), whereas random pairs should have a small probability (close to 0). There is a last modeling question that needs to be addressed:

Q What is a good choice for the distribution q used to sample negative word pairs?

In principle, one could just use the relative frequencies $p(w)$ of words to define the negative sampling distribution q. This would ensure that we ignore the conditioning information and (in the limit) reproduce the empirical marginals. However, based on experiments it has been shown to be beneficial to slightly generalize this to the choice

$$q(w) \propto p(w)^\alpha, \quad \alpha \geq 0 \tag{3.48}$$

where $p(w)$ is the relative frequency of the word w in the corpus. A typical exponent used in practice is $\alpha = 3/4$, which slightly over-samples infrequent words. The intuition behind this is that what matters most in learning semantic representations is not the very frequent words – which often carry little meaning – and also not the very infrequent words, but the in-between range.

[4]Note that one may randomly select words that are actually occurring in the context window. Such conflicts, leading to some amount of label noise, are ignored, as they do not make much of a difference in practice.

3.3.4 Pointwise Mutual Information

Q How can we further justify and understand the negative sampling approach?

Let us denote by $p(v,w)$ the true distribution of co-occurring words and by $q(v,w) = p(w)q(v)$ the distribution used for negative sampling. What would be the optimal classifier that could be achieved by the embedding model? It is the so-called optimal Bayesian classifier. It can be written as

$$\mathbb{P}((v,w) = \text{true}) = \frac{\pi p(v,w)}{\pi p(v,w) + (1-\pi)q(v,w)} \tag{3.49}$$

where π and $1-\pi$ are the prior probabilities of true and false (i.e. random) pairs. If we consider the pre-image under the logistic function, some easy calculations shows

$$h^*_{vw} = \sigma^{-1}\left(\frac{\pi p(v,w)}{\pi p(v,w) + (1-\pi)q(v,w)}\right) = \ln\frac{p(v,w)}{q(v,w)} + \ln\frac{\pi}{1-\pi} \tag{3.50}$$

So, in the case of balanced classes $\pi = \frac{1}{2}$ and $\alpha = 1$ we get

$$h^*_{vw} = \ln\frac{p(v,w)}{p(w)q(v)} \tag{3.51}$$

which can be identified as the contribution of the pair (v,w) to the mutual information, the so-called *point-wise* mutual information.

→ The word2vec approach with balanced negative sampling (and $\alpha = 1$) can be interpreted as a method to maximize the mutual information of word co-occurrences.

3.3.5 GloVe and Matrix Factorization

Q Can we think of word embedding models for co-occurrences in terms of matrix factorization?

We now aim to connect word embeddings to the topic of matrix factorization. Let us consider the co-occurrence counts and the square matrix formed by them

$$N_{ij} := \{(t,t') : x_t = w_i,\ x_{t'} = w_j,\ (t'-t) \in \mathcal{I}\} \tag{3.52}$$

We can use the convention $\ln(0) = 0$ and then consider the matrix of log-counts. The so-called GloVe (global word vector) objective is given by

$$\ell(\boldsymbol{\theta}, \mathbf{N}) = \sum_{v,w:N_{vw}>0} f(N_{vw})\left[\ln N_{vw} - \ln \hat{N}_{vw}\right]^2, \quad \hat{N}_{vw} = \tilde{p}(v,w;\boldsymbol{\theta}) \tag{3.53}$$

which is a weighted squared loss on the log-scale. The weighting function f is a design choice. A choice that has worked well in practice is

$$f(N) = \min\left\{1, \left(\frac{N}{N_{\max}}\right)^\alpha\right\}, \quad \text{e.g. } \alpha = 3/4\,. \tag{3.54}$$

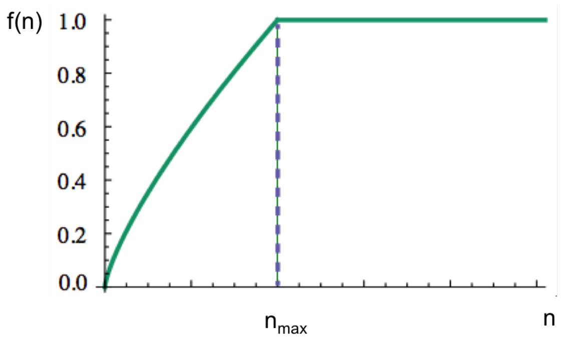


Figure 3.12: Weighting function used in GloVe

See the plot in Figure 3.12. Now the clue with this objective is that we can work with an unnormalized conditional probability distribution and simply chose

$$\ln \hat{p}(v, w) = \langle \boldsymbol{\zeta}_w, \mathbf{z}_v \rangle \tag{3.55}$$

without any normalization constant.

$\rightarrow$ With the GloVe objective, one can work with unnormalized probability models.

The reason is that the squared objective is two-sided and will not degenerate without normalization, whereas in a likelihood-based criterion, increasing probabilities is always better and the balancing effect comes purely from the normalization of the probability mass function (not all events can have high probability as they “compete" for probability mass). As a final remark, one can also introduce bias terms, but can absorb these into the embeddings, e.g. by clamping $\zeta_{w1} = 1$ and $z_{w2} = 1$ $(\forall w)$, say. Then ζ_{w2} and z_{w1} take the role of bias parameters.

The unnormalized view allows us to directly interpret the model in terms of matrix factorization. Define with $n = |\Sigma|$

$$\mathbf{U}^\top := [\boldsymbol{\zeta}_{w_1} \cdots \boldsymbol{\zeta}_{w_n}], \quad \mathbf{V} := [\mathbf{z}_{w_1} \cdots \mathbf{z}_{w_m}], \quad \text{then} \quad \ln \hat{N} = \mathbf{UV}\,. \tag{3.56}$$

The GloVe objective is a weighted Frobenius norm of the approximation residual between the observed log-count matrix and a low-rank factorization of embedding matrices. Also note that the use of two different embeddings is very natural in the context of matrix factorizations.

$\rightarrow$ The GloVe objective with a bi-linear embedding model can be thought of in terms of a low-rank matrix factorization.

As a special case consider the weighting function

$$f(N) = \min\{1, N\}\,. \tag{3.57}$$

It results in a matrix completion problem

$$\mathbf{U}, \mathbf{V} \overset{\min}{\rightarrow} \sum_{i,j:N_{ij}>0} (\ln N_{ij} - (\mathbf{UV})_{ij})^2 \tag{3.58}$$

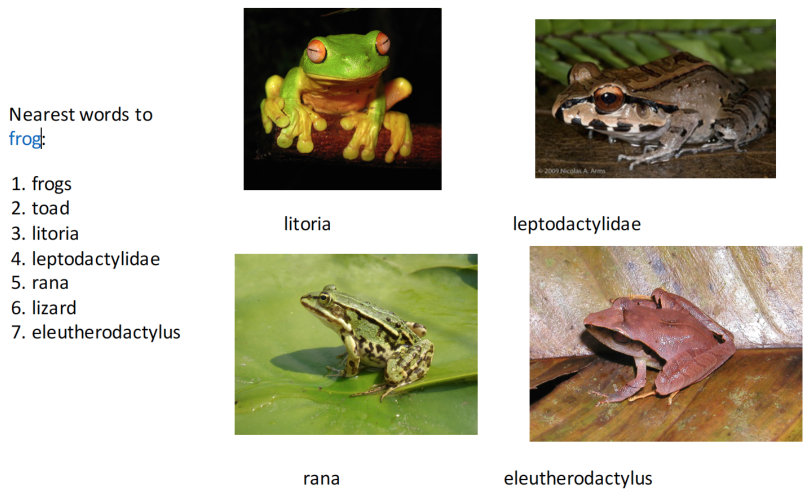


Figure 3.13: Nearest word to frog in a GloVe model along with illustrating photos.

As is clear from the view of GloVe as a matrix factorization problem, we have to resort to approximation techniques to find a solution. There are a variety of options. We present here a stochastic gradient approach that is particularly appealing. Sample a pair (v, w) such that $N_{vw} > 0$ at random. Then perform updates with a step size $\eta > 0$ as follows

$$\boldsymbol{\zeta}_w \leftarrow \boldsymbol{\zeta}_w + 2\eta f(N_{vw}) \left(\ln N_{vw} - \langle \boldsymbol{\zeta}_w, \mathbf{z}_v \rangle\right) \mathbf{z}_v \tag{3.59}$$

$$\mathbf{z}_v \leftarrow \mathbf{z}_v + 2\eta f(N_{vw}) \left(\ln N_{vw} - \langle \boldsymbol{\zeta}_w, \mathbf{z}_v \rangle\right) \boldsymbol{\zeta}_w \tag{3.60}$$

One simple approximate optimization method to learn word embeddings is stochastic gradient descent.

3.3.6 Discussion

Let us conclude this section by providing some examples of results and a brief discussion. One can investigate word embeddings with regard to how well they capture semantic similarity. An example for GloVe embeddings is shown in Figure 3.13. It is also well-known that one can exploit the affine structure to solve word analogy problems. For instance king is to man what w is to woman can be solves by

$$w = \arg\max_v \langle \boldsymbol{\zeta}_{\text{king}} - \boldsymbol{\zeta}_{\text{man}} + \boldsymbol{\zeta}_{\text{woman}}, \boldsymbol{\zeta}_v \rangle \tag{3.61}$$

This affine structure can also be visualized in plots like shown in Figure 3.14.

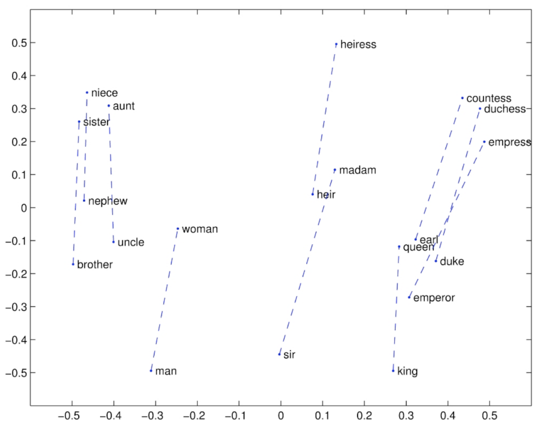


Figure 3.14: 2d projection of word embeddings.

4. Deep Neural Networks

4.1 Compositional Models

4.1.1 Power of Compositionality

Q How can we build powerful models out of simple building blocks?

The world is complex. Building accurate models requires a powerful and flexible approach to model design. We will consider here the general problem of learning an unknown vector-valued function or map $\Psi : \mathbb{R}^n \to \mathbb{R}^p$ or simply a function $\psi : \mathbb{R}^n \to \mathbb{R}$. There are three general approaches to model design philosophies developed in the area of machine learning.

Feature Engineering

Feature Engineering relies on the use of domain knowledge to develop features, which make relevant information explicit. In such representations, simple models such as a linear or generalized linear ones are often sufficient to solve the task in question. Thus the role played by learning from data is minimal. Feature engineering is applicable with very little data, but is limited by requiring human ingenuity and effort in engineering suitable features.

Q How can we avoid the need for hand-crafted features?

Expansive Representations

A second strategy uses the idea of feature *expansion*: one first maps the input to a higher-dimensional representation $H : \mathbb{R}^n \to \mathbb{R}^p$, often with $p \gg n$ and then uses an adaptable linear map g to get $\psi = g \circ H$. The case where H is non-adaptable leads to important methods such as kernel machines and Support Vector Machines (SVMs). The emphasis is on how to obtain large – even infinite – dimensionality p, while retaining a tractable learning algorithm and generalization capabilities. The idea of expansion is also used in neural networks when considering wide layers (see below). It is clear that the more feature we extract, the more powerful the resulting function.

→ Feature expansion provides a basic principle to increase the complexity of model classes.

Compositional Representation

An alternative – even more powerful – strategy is at the core of what has spurred enormous progress in the last 10 years: the use of *compositionality*. Instead of expanding once into a (fixed) high-dimensional feature representation, complexity is achieved and managed through levels of composition or *depth*. We will first focus on what is known as a feedforward network:

$$\psi = g \circ \underbrace{H_L \circ H_{L-1} \circ \cdots \circ H_2 \circ H_1}_{\text{depth } L}, \quad H_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}, \; n_0 = n \,. \tag{4.1}$$

We can also write this with argument in nested bracket form

$$\psi(\mathbf{x}) = g(H_L(\ldots H_2(H_1(\mathbf{x}))\ldots)) \tag{4.2}$$

We refer to each map H_l as a *layer map*. Note that layers can be expansive $n_l > n_{l-1}$ or non-expansive $n_l \leq n_{l-1}$. The compositional structure in Eq. (4.1) implies that downstream maps operate on the output of upstream maps. The partial maps $H_{l:1} := H_l \circ \cdots \circ H_1$ produce *intermediate representations*. Due to the sequential nature of processing (Markov property), these representations need to preserve task-relevant information about the input, while making it more accessible and explicit with increasing depth l. The intuition behind this can be illustrated with the example of object recognition in computer vision, where we expect a DNN to extract more semantic, high-level features with depth (cf. Figure 4.1).

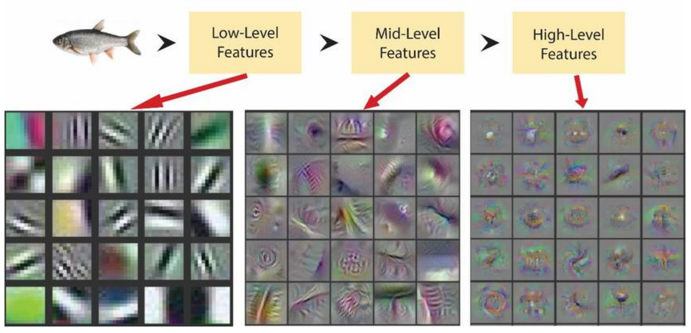


Figure 4.1: Visualization of intermediate feature representations.

→ Deep neural network combine compositionality (depth) with expansiveness (width) to learn suitable signal representations.

Q How can we define adaptable layer maps to be composed in a deep neural network?

4.1.2 Ridge Functions

Arguably the simplest family of multivariate maps are linear maps that can be (finitely) parameterized by a weight matrix

$$F : \mathbb{R}^n(\times\mathbb{R}^{m\times n}) \to \mathbb{R}^m, \quad F(\mathbf{x};\boldsymbol{\Theta}) = \boldsymbol{\Theta}\mathbf{x}, \quad \boldsymbol{\Theta} \in \mathbb{R}^{m\times n} \tag{4.3}$$

Each component function of F represents a linear function, which is a weighted combination of the inputs (and hence the name *weight* matrix). It is easy to see, however, that linear maps are closed under composition and hence we cannot increase their modeling power through composition.

Proposition 4.1.1 Let F and G be linear maps. Then $H = G \circ F$ is linear.

Proof. Let $\mathbf{A}$ be the matrix representing F and $\mathbf{B}$ the matrix representing G, then

$$H(\mathbf{x}) = (G \circ F)(\mathbf{x}) = \mathbf{B}\mathbf{A}\mathbf{x} = \mathbf{C}\mathbf{x}, \quad \text{where} \quad \mathbf{C} = \mathbf{B}\mathbf{A}$$

■

Q As linear maps are not suitable for compositional models. What is the simplest extension of linear maps that unlocks compositionality?

Obviously we need to introduce non-linearity somewhere. It would be desirable to stick with weight matrices and to not introduce additional (possibly complicated) parameterizations for the non-linear part. Hence we could consider maps $\Phi \circ F$, where F is a parameterized linear map and $\Phi : \mathbb{R}^m \to \mathbb{R}^m$ is a fixed non-linear map. Following a minimalistic philosophy, it seems appealing to define Φ independently per dimension as follows

$$\Phi(\mathbf{x}) = \begin{pmatrix} \phi(x_1) \\ \dots \\ \phi(x_m) \end{pmatrix}, \quad \phi : \mathbb{R} \to \mathbb{R} \tag{4.4}$$

This approach has the advantage to work for any dimensionality m. It is minimalistic in that we only need to define a single non-linearity ϕ, also known as an *activation* function. The resulting functions and maps are sometimes also called *ridge functions*. Note that the component functions takes the simple form

$$f(\mathbf{x}) = \phi(\langle \boldsymbol{\theta}, \mathbf{x} \rangle) \tag{4.5}$$

with adjustable weights $\boldsymbol{\theta}$. We also call such an elementary function a *unit* or computational neuron. The analogy to nervous systems is also where the name neural networks originates from.

$\rightarrow$ Ridge functions with fixed activation functions provide the most minimalistic approach to generalize linear maps.

Minimalistic views have appeal, but one needs to make sure that they accomplish the goals pursued. There is a research area dealing with questions of approximation power of representation systems for functions. One of the classic fundamental results is that neural networks with two composed layer maps (e.g. with one so-called intermediate or *hidden layer*) are sufficient to approximate any continious function to a given (but arbitrary) accuracy as long as one can increase the hidden layer width sufficiently and as long as ϕ is *not* a polynomial.

→ Neural networks with one hidden layer and a non-polynomial activation function are universal function approximators.

This result is very clean, yet also coarse in that it does not differentiate between different architectures and their compositionality. Recent work in the area of Deep Learning has aimed to highlight the advantages of depth, for instance. This is, however, well-beyond the scope of this lecture.

4.1.3 Sigmoid and Rectified Layers

For concreteness, let us consider the two most popular activation functions.

Sigmoid Layers

The first, classical choice is a sigmoid activation function, often chosen to be the logistic function or the hyperbolic tangent

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \tanh(z) = 2\sigma(2z) - 1 \tag{4.6}$$

This has often been motivated as a soft-threshold function, which also incorporates the biologically motivated idea of *saturation*. Layers or networks with such activation functions are often called sigmoid layers or networks, respectively. Logistic units are also often used as output activations, when implementing a probabilistic binary classifier. This generalizes logistic regression.

→ The choice of a logistic activation function leads to sigmoid networks, also used in the classical multi-layer perceptron (popular in the second wave of neural networks started in 1986, called *connectionism*).

Piecewise Linear Layers

The second, more modern choice is the rectified linear unit (ReLU) or soft-plus function,

$$h : \mathbb{R}^n \to \mathbb{R}, \quad h(\mathbf{x}) = (\langle \boldsymbol{\theta}, \mathbf{x} \rangle)_+ \tag{4.7}$$

$$(z)_+ = \max\{0, z\} \tag{4.8}$$

This activation function consists of two linear pieces. If we combine it with a linear function, then the input space is partitioned into two halfspaces

$$\mathcal{X}^+ = \{\mathbf{x} : \langle \mathbf{w}, \mathbf{x} \rangle > 0\}, \quad \mathcal{X}^- = \mathbb{R}^m - \mathcal{X}^+, \tag{4.9}$$

such that the unit is linear on $\mathcal{X}^+$ and constantly 0 on $\mathcal{X}^-$.

→ Rectified layers are used in many modern architectures and are linear on a halfspace of its inputs, zero on the complementary halfspace.

4.1.4 Classical MLP

Let us discuss a concrete example of a DNN to sharpen our understanding, before we move to a more abstract level. Possibly the single most important neural network architecture is the 3-layer MLP (multi-layer percetron): a true rock star! This reference architecture for n

inputs and a single output has a hidden layer with m sigmoid units. We can easily write down the modeled function as

$$\psi(\mathbf{x}; \boldsymbol{\beta}; \boldsymbol{\Theta}) = \sum_{j=1}^{m} \frac{\beta_j}{1 + \exp[-\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle]} \tag{MLP}$$

Each unit computes a specific weighted combination of input features and applies a logistic activation function. These activations (in range $(0;1)$) are then summed up with weights $\boldsymbol{\beta}$.

→ The MLP is the classical reference architecture for neural network.

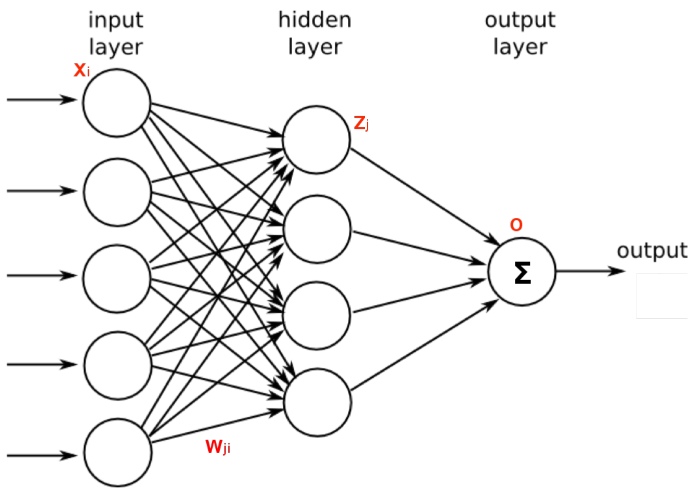


Figure 4.2: Diagrammtic sketch of a 3 layer MLP.

Let us see how we can compute gradients for an MLP that can be used to tune its parameters. Assuming a squared loss, elementary calculations yield

$$\frac{\partial \frac{1}{2}(\psi(\mathbf{x}) - y)^2}{\partial \beta_j} = \frac{\psi(\mathbf{x}) - y}{1 + \exp[-\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle]} \tag{4.10}$$

$$\frac{\partial \frac{1}{2}(\psi(\mathbf{x}) - y)^2}{\partial \theta_{ji}} = \frac{\psi(\mathbf{x}) - y}{1 + \exp[-\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle]} \cdot \frac{\beta_j x_i}{1 + \exp[\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle]} \tag{4.11}$$

Let us derive the somewhat more involved second part in small steps.

$$\frac{\partial \frac{1}{2}(\psi(\mathbf{x}) - y)^2}{\partial \theta_{ji}} = \underbrace{(\psi(\mathbf{x}) - y)}_{\text{residual } \delta} \frac{\partial \psi(\mathbf{x})}{\partial \theta_{ji}} \tag{4.12}$$

$$\frac{\partial \psi(\mathbf{x})}{\partial \theta_{ji}} = \beta_j \frac{\partial \sigma(\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle)}{\partial \theta_{ji}} = \beta_j x_i \underbrace{\sigma'(\langle \boldsymbol{\theta}_j, \mathbf{x} \rangle)}_{\substack{\text{derivative of} \\ \text{logistic function}}} \tag{4.13}$$

The calculation concludes by exploiting the easily checkable fact that

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) = \sigma(z)\sigma(-z) \tag{4.14}$$

Learning then typically proceeds via Stochastic Gradient Descent via selecting a random minibatch $\mathcal{S}_t$ from the training data $\mathcal{S}$ at every iteration t. We then calculate a (random) update direction as

$$\triangle_\vartheta(\boldsymbol{\beta},\boldsymbol{\Theta}) := \sum_{(\mathbf{x},y)\in\mathcal{S}_t} \frac{\partial \frac{1}{2}(\psi(\mathbf{x};\boldsymbol{\beta},\boldsymbol{\Theta}) - y)^2}{\partial\vartheta}, \quad \vartheta \in \{\beta_i, \theta_{ji}\} \tag{4.15}$$

$$\vartheta_{t+1} = \vartheta_t - \eta\, \triangle_\vartheta(\boldsymbol{\beta}_t,\boldsymbol{\Theta}_t) \tag{4.16}$$

where $\eta > 0$ is a suitably chosen step size parameter. The cardinality $B = |\mathcal{S}_t|$ is called the mini-batch size. $B = 1$ corresponds to the classical SGD algorithm. Note that the partial derivative of each specific parameter ϑ depends on all other parameters, i.e. $(\boldsymbol{\beta},\boldsymbol{\Theta})$.

Implementing an MLP and vanilla gradient-based learning (where $\mathcal{S}_t = \mathcal{S}$, for simplicity) requires just a few lines of code.

```
class MLP3():
    def __init__(self,n,m):
        self.theta = np.random.uniform(-1/n,1/n,(m,n)) # mxn
        self.beta  = np.random.uniform(-1/m,1/m,(1,m)) # 1xm
        self.n     = n
        self.m     = m
```

The forward propagation can be implemented as follows:

```
    def forward(self, x):
        f         = {}
        net_in    = np.dot(self.theta,x)        # mxn*nxs=mxs
        f['hid']  = expit(net_in)               # mxs
        f['out']  = np.dot(self.beta,f['hid']) # 1xm*mxs=1xs
        return f
```

where we retain the hidden layer activities. We can then compute gradients as follows

```
    def gradient(self, f, x, y):
        g         = {}                          # beta gradients
        delta     = (f['out']-y)                # 1xs
        g['bet']  = np.dot(delta,f['hid'].T)    # 1xs*sxm=1xm
                           # theta gradients
        hid       = f['hid']
        hid_sv    = hid-hid*hid                 # sensitivities
        outer     = np.outer(self.beta.T,delta) # mx1*1xs=mxs
        g['tet']  = np.dot((outer*hid_sv),x.T)  # mxs*sxn=mxn
        return g
```

Compared to Eqs. (4.10),(4.11) we want to be able to work with data matrices. This introduces another dimension of size s (=number of samples). In the final gradient computation, we can reduce over this dimension by summing. We can then perform (batch) gradient descent via

```
for k in range(steps):
    f           = model.forward(X)
    g           = model.gradient(f,X.Y)
    model.beta  = model.beta  - (eta/s)*g['bet']
    model.theta = model.theta - (eta/s)*g['tet']
```

→ Implementing gradient-based MLP learning just involves a few lines of code. Deriving gradients is elementary.

4.2 Backpropagation

Q How can deep neural network be generically trained efficiently?

Gradient-based methods like SGD are very – perhaps even unreasonably – effective in training DNNs. This holds for training accuracy, but – more importantly – also for generalization accuracy. We take this as given here and focus on algorithmic aspects. So the above question is largely tantamount to the question of how to perform gradient descent, which foremost involves the computation of gradients in computational models.

4.2.1 Single Unit

Q What is the error-gradient of the parameters of a single unit?

Consider a single (generic) unit h with associated parameters $\boldsymbol{\theta}$, $h = \phi(\langle \boldsymbol{\theta}, \mathbf{z} \rangle)$. Assume that we can calculate the partial derivative of the loss with regard to h (the unit's activation),

$$\delta := \frac{\partial \ell}{\partial h}, \tag{4.17}$$

which is also called the *error signal* or *delta* for a unit. Intuitively this quantifies how changing the activation of the unit influences the loss (in an infinitesimal sense). Note that during learning, $\mathbf{z}$ is fixed as it is simply an upstream function of the input $\mathbf{x}$, obtained by forward propagating through the network. However, we are interested in the effect of changing the parameter, which by the chain rule is given by

$$\nabla_{\boldsymbol{\theta}} \ell = \frac{\partial \ell}{\partial h} \nabla_{\boldsymbol{\theta}} h = \delta \, \phi'(\langle \boldsymbol{\theta}, \mathbf{z} \rangle) \, \mathbf{z} \tag{4.18}$$

This is very intuitive:

→ The local parameter gradient is the product of an upstream vector, a (scalar) downstream error signal, and the local sensitivity of the unit.

Q How can we compute error signals efficiently?

4.2.2 Jacobi Map Recurrence

The only remarkable thing about error backpropagation is how to organize the collective computation of error signals for all units. Let us switch to a layer-map view. Below we state an elementary recurrence, which follows directly from the chain rule (for $k < L$)

$$\boldsymbol{\delta}_k := \frac{\partial \ell}{\partial H_k} = \left[\frac{\partial H_{k+1}}{\partial H_k} \right]^\top \cdot \frac{\partial \ell}{\partial H_{k+1}} =: \mathbf{J}_{k+1}^\top \cdot \boldsymbol{\delta}_{k+1} = \mathbf{J}_{k+1}^\top \cdot \ldots \cdot \mathbf{J}_L^\top \cdot \boldsymbol{\delta}_L \tag{4.19}$$

where the appearing matrices are the (transposed) Jacobi matrices of the respective layer maps. The Jacobi matrix for H_k conveniently summarizing all partial derivatives of the layer (output) activations with regard to the layer input (activations). For a map H it is defined as:

$$H : \mathbb{R}^n \to \mathbb{R}^m, \quad H(\mathbf{z}) = \begin{pmatrix} h_1(\mathbf{z}) \\ \dots \\ h_m(\mathbf{z}) \end{pmatrix}, \quad [\mathbf{J}_H]_{ij} := \frac{\partial h_i}{\partial z_j} \tag{4.20}$$

“Convenient" here means in particular that the Jacobi matrix of a composed map can be written as a product of the Jacobi matrices of the elementary maps, which is exploited in the above formula. It is important to stress that the Jacobi ‘matrix’ is – in reality – a matrix-valued map. This, of course, follows from the fact that each partial derivative is a function, which can be evaluated at an argument. The argument in question is implicit in the hidden layer activation that are deterministically induced by an input. So every $\mathbf{J}_l$ depends on an activation $\mathbf{z}_{l-1}$, which in turn depends on $\mathbf{x}$. Note that $\boldsymbol{\delta}_L$ will be a function of an input-target pair $(\mathbf{x}, y)$. So the forward pass will determine all $\mathbf{J}_l$ as well as $\boldsymbol{\delta}_L$ and then a backward pass sequentially calculates $\boldsymbol{\delta}_{L-1}, \dots \boldsymbol{\delta}_1$ in reverse order.

→ All error signals can be calculated in backward order through vector-matrix multiplications with Jacobi matrices determined in the forward pass.

4.2.3 Losses and Error Signals

For the sake of concreteness, we simply derive the initial error signals for the the two most popular loss functions. First consider the case of the squared loss, where ones has

$$\ell(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(\mathbf{y} - \Psi(\mathbf{x}))^2, \quad \boldsymbol{\delta}_L = \frac{\partial \ell}{\partial H_L} = \Psi(\mathbf{x}) - \mathbf{y}\,, \tag{4.21}$$

which means that the backpropagation recurrence is initialized with the residual.

As a second special case, consider the logistic loss for $y \in \{-1, 1\}$ and $n_L = 1$

$$\ell(\mathbf{x}, y) = -\ln \sigma(y\psi(\mathbf{x})), \quad \delta_L = \frac{\partial \ell}{\partial h_L} = -y\sigma(-y\psi(\mathbf{x})) \tag{4.22}$$

4.2.4 Jacobi Matrices

Let us also shed some light on the Jacobi matrices for special choices of activation functions. First notive that for linear layers with $\phi = \text{id}$ we simply get

$$\mathbf{J}_H = \boldsymbol{\Theta} \tag{4.23}$$

as the linearization of a linear map is just the map itself.

In the case of a ReLU layer, we get

$$\mathbf{J}_H = \text{diag}(\boldsymbol{\chi})\, \boldsymbol{\Theta}, \quad \boldsymbol{\chi} \in \{0,1\}^m, \quad \boldsymbol{\chi} = \mathbb{I}[\boldsymbol{\Theta}\mathbf{z} > 0]\,. \tag{4.24}$$

This can be interpreted as follows: the Boolean variables χ_j indicate whether a unit is active or not (given the current input). Inactive units are effectively removed in the backpropagation step. For active units, the result agrees with the linear case. Conceptually, we can also think of a ReLU as follows: given an input, prune all inactive units. The pruned networks is just a linear DNN.

Finally, let us consider the case of sigmoid DNNs. Here we get an additional diagonal matrix with unit sensitivities:

$$\mathbf{J}_H = \text{diag}(\boldsymbol{\chi})\, \boldsymbol{\Theta}, \quad \boldsymbol{\chi} \in (0,1)^m, \quad \boldsymbol{\chi} = \sigma'(\boldsymbol{\Theta}\mathbf{z}) \tag{4.25}$$

4.3 Gradient Descent

The state-of-the-art methods for training DNNs are almost all variants of gradient descent. We would like to collect some basic results here, which are entry points to a much richer literature that continues to evolve and mature. Let $\ell : \mathbb{R}^d \to \mathbb{R}$ be a differentiable objective function and define the iterate sequence generated by gradient descent as

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \nabla \ell(\boldsymbol{\theta}^k), \quad \eta > 0, \tag{4.26}$$

where $\boldsymbol{\theta}^0$ is a starting point. A very fundamental question is the following:

Q Will gradient descent converge to an optimal solution and if so at what rate?

4.3.1 Convex Quadratics

Let us first consider the case of quadratic problems, which is *not* what is needed for DNNs, but it is the simplest problem to study. Moreover, it allows for an exact analysis that can inspire more general problems. Let us thus start with objectives of the type

$$\ell(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{Q}\, \boldsymbol{\theta} - \mathbf{q}^\top \boldsymbol{\theta}, \quad \mathbf{Q} : \text{positive definite} \tag{4.27}$$

Q How can we characterize the (asymptotic) behavior of gradient descent on a convex quadratic objective?

Let us diagonalize $\mathbf{Q} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$ with orthogonal $\mathbf{U}$. Then we can perform a change of basis $\boldsymbol{\theta} \leftarrow \mathbf{U}^\top\boldsymbol{\theta}$, $\mathbf{q} \leftarrow \mathbf{U}^\top\mathbf{q}$ and effectively minimize the separable problem

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^d \ell_i(\theta_i), \quad \ell_i(\vartheta) = \frac{\lambda_i}{2}\vartheta^2 - q_i\vartheta, \quad \lambda_i > 0\,. \tag{4.28}$$

The derivatives are $\ell_i'(\vartheta) = \lambda_i\vartheta - q_i$, which leads to the first order optimality condition

$$\theta_i^* = \frac{q_i}{\lambda_i}, \quad \ell_i^* := \ell_i(\theta_i^*) = -\frac{q_i^2}{2\lambda_i} \tag{4.29}$$

In order to further simplify the analysis, let us shift each ℓ_i so that $\min \ell_i = 0$,

$$\ell_i(\vartheta) = \frac{1}{2\lambda_i}\left(\lambda_i\vartheta - q_i\right)^2 \tag{4.30}$$

which completes the square. A gradient step results in

$$\ell_i(\vartheta - \eta(\lambda_i\vartheta - q_i)) = \frac{1}{2\lambda_i}\left((1-\lambda_i\eta)(\lambda_i\vartheta - q_i)\right)^2 = (1-\lambda_i\eta)^2\ell_i(\vartheta) \tag{4.31}$$

As long as $\eta < 2/\lambda_i$ the objective decreases by a factor of less than 1 in every step. Coming back to Eq. (4.27) we see that the condition that needs to be imposed on the step size is

$$\eta < \frac{2}{\lambda_{\max}} \tag{4.32}$$

where $\lambda_{\max}$ is the largest eigenvalue of $\mathbf{Q}$.

→ With appropriately chosen step size, gradient descent converges exponentially fast to the minimum of a convex quadratic.

If we want to optimize the rate of convergence, we would best chose

$$\eta^* = \arg\min_\eta \max_i (1-\eta\lambda_i)^2 = \arg\min_\eta \max\{\eta\lambda_{\max}-1, 1-\eta\lambda_{min}\} \tag{4.33}$$

which is attained at

$$\eta\lambda_{\max} - 1 \overset{!}{=} 1-\eta\lambda_{\min} \iff \eta^* = \frac{2}{\lambda_{\max}+\lambda_{\min}} \tag{4.34}$$

and results in the bets rate of

$$\rho = (1-\lambda_{\min}\eta^*)^2 = \left(\frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}}\right)^2 \le \left(\frac{\kappa-1}{\kappa+1}\right)^2, \quad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{4.35}$$

Here κ is the condition number of $\mathbf{Q}$, which appears after dividing numerator and denominator by $\lambda_{\min}$.

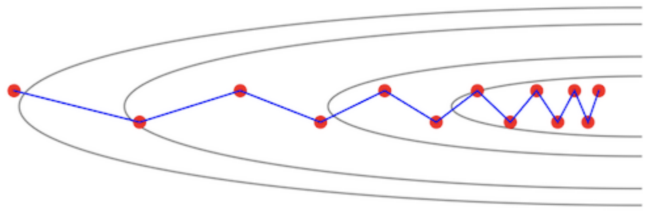


Figure 4.3: Quadratic with $\kappa = 10$ and the gradient descent iterates.

4.3.2 Smoothness

Q How can we ensure gradient information remains informative in a neighborhood around a point?

Fundamentally, it is clear that a local method like gradient descent can only work reliably, if the gradient does not change too much relative to the step size taken. The typical condition to ensure this is to require some degree of smoothness.

Definition 4.3.1 — Smooth Function. A differentiable function $\ell : \mathbb{R}^d \to \mathbb{R}$ is L-smooth for some $L > 0$, if

$$\|\nabla\ell(\boldsymbol{\theta}) - \nabla\ell(\boldsymbol{\vartheta})\| \le L\|\boldsymbol{\theta}-\boldsymbol{\vartheta}\| \quad (\forall\, \boldsymbol{\theta},\, \boldsymbol{\vartheta})$$

Smoothness of ℓ simply means that its gradient function $\nabla\ell$ is Lipschitz continuous with constant L: within an ϵ-ball around any point the change of gradient cannot amount to a norm difference of more than ϵL.

There are immediate implication of smoothness. For simplicity of argument, let us assume that ℓ is twice (continuously) differentiable. We can apply Taylor's theorem with integral remainder and with some (unspecified) $\boldsymbol{\beta}$ arrive at

$$\begin{aligned}\ell(\boldsymbol{\vartheta}) - \ell(\boldsymbol{\theta}) &= \langle\nabla\ell(\boldsymbol{\theta}), \boldsymbol{\vartheta}-\boldsymbol{\theta}\rangle + \frac{1}{2}(\boldsymbol{\vartheta}-\boldsymbol{\theta})^\top\nabla^2\ell(\boldsymbol{\beta})(\boldsymbol{\vartheta}-\boldsymbol{\theta}) \\ &\le \langle\nabla\ell(\boldsymbol{\theta}), \boldsymbol{\vartheta}-\boldsymbol{\theta}\rangle + \frac{L}{2}\|\boldsymbol{\vartheta}-\boldsymbol{\theta}\|^2 \end{aligned} \tag{4.36}$$

In the special case of gradient descent we have $\boldsymbol{\vartheta} = \boldsymbol{\theta} - \eta \nabla \ell(\boldsymbol{\theta})$ and thus

$$\ell(\boldsymbol{\vartheta}) - \ell(\boldsymbol{\theta}) \le -\eta \left(1 - \frac{L\eta}{2}\right) \|\nabla \ell(\boldsymbol{\theta})\|^2 \tag{4.37}$$

which leads to a guaranteed decrease, if $\eta < 2/L$ and to the optimal choice (relative to the applied bound) of $\eta = 1/L$, im which case one gets

$$\ell\left(\boldsymbol{\theta} - \tfrac{1}{L}\nabla \ell(\boldsymbol{\theta})\right) - \ell(\boldsymbol{\theta}) \le -\frac{1}{2L}\|\nabla \ell(\boldsymbol{\theta})\|^2 . \tag{4.38}$$

By identifying L and $\lambda_{\max}$, this is a natural generalization of the quadratic case.

The other relevant quantity on which the success of gradient descent hinges is the gradient norm. If it vanishes (i.e. becomes too small too quickly as we approach a minimum), then convergence may become prohibitively slow. One way to circumvent this problem is to ask how quickly we reach a point of small gradient, assuming, this is in itself a good (enough) solution.

Definition 4.3.2 — ϵ-Critical Point. Let ℓ be differentiable at $\boldsymbol{\theta}$, then $\boldsymbol{\theta}$ is an ϵ-critical point, if $\|\nabla \ell(\boldsymbol{\theta})\| \le \epsilon$.

We then get

Theorem 4.3.1 Gradient descent on an L-smooth, differentiable function ℓ finds an ϵ-critical point in at most $k = 2L(\ell(\boldsymbol{\theta}^0) - \ell^*)/\epsilon^2$ steps.

Proof. Note that with $C := \ell(\boldsymbol{\theta}^0) - \ell^* \ge 0$

$$-C \le \ell(\boldsymbol{\theta}^k) - \ell(\boldsymbol{\theta}^0) \le -\frac{1}{2L}\sum_{r=0}^{k-1} \|\nabla \ell(\boldsymbol{\theta}^r)\|^2 \iff \frac{1}{k}\sum_{r=0}^{k-1} \|\nabla \ell(\boldsymbol{\theta}^r)\|^2 \le \frac{2LC}{k}$$

This means that for at least one of the iterates $\boldsymbol{\vartheta} \in \{\boldsymbol{\theta}^0, \dots, \boldsymbol{\theta}^{k-1}\}$ we have that

$$\|\ell(\boldsymbol{\vartheta})\|^2 \le \frac{2LC}{k} \overset{!}{\le} \epsilon^2 \iff k \ge \frac{2LC}{\epsilon^2}$$

■

Practically speaking, as we calculate the gradients, we can easily check the gradient norm condition and stop at the desired ϵ.

→ Smoothness is sufficient to find ϵ-critical points with $\mathbf{O}(\epsilon^{-2})$ steps of gradient descent.

4.3.3 Strong Convexity and the PL-Condition

Q What further conditions can guarantee (fast) convergence to a local minimum?

The key idea is to tie the gradient norm to the suboptimality of the solution. A classical notion of how to do this is known as the PL-condition.

Definition 4.3.3 — Polyak-Łojasiewicz Condition. A differentiable function ℓ obeys the Polyak-Łojasiewicz Condition with parameter $\mu > 0$, if and only if

$$\frac{1}{2}\|\nabla\ell(\boldsymbol{\theta})\|^2 \geq \mu\left(\ell(\boldsymbol{\theta}) - \ell^*\right) \ \ (\forall\boldsymbol{\theta}), \quad \ell^* = \min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$$

The power of this condition is illustrated in the following theorem where the PL-condition directly results in a geometric convergence guarantee.

Theorem 4.3.2 Let ℓ be differentiable, L-smooth and μ-PL. Then gradient descent with step size $\eta = 1/L$ converges at a geometric rate

$$\ell(\boldsymbol{\theta}^k) - \ell^* \leq \left(1 - \frac{\mu}{L}\right)^k \left(\ell(\boldsymbol{\theta}^0) - \ell^*\right).$$

Proof.

1. From Eq. (4.38)
$$\ell(\boldsymbol{\theta}^{k+1}) - \ell(\boldsymbol{\theta}^k) \leq -\frac{1}{2L}\|\nabla\ell(\boldsymbol{\theta}^k)\|^2$$
2. From the PL condition
$$-\|\nabla\ell(\boldsymbol{\theta}^k)\|^2 \leq -2\mu(\ell(\boldsymbol{\theta}^k) - \ell^*)$$
3. Subtracting ℓ^* yields
$$\ell(\boldsymbol{\theta}^{k+1}) - \ell^* \leq \left(1 - \frac{\mu}{L}\right)\left(\ell(\boldsymbol{\theta}^k) - \ell^*\right)$$
4. The claim follows by induction.

■

For simplicity, we analyzed $\eta = 1/L$, but note that L-smoothness implies L'-smoothness for any $L' \geq L$ and thus the result applies to any $\eta \leq 1/L$.

→ The PL condition is a fundamental property that (combined with smoothness) directly implies geometric convergence to the minimum.

Q What are well-known functions that obey the PL-condition?

When ℓ is convex, the PL-condition is implied by strengthening the convexity condition by a lower quadratic bound.

Definition 4.3.4 — Strongly Convex Function. A differentiable function ℓ is μ-strongly convex for some $\mu > 0$, if it fulfills

$$\ell(\boldsymbol{\vartheta}) \geq \ell(\boldsymbol{\theta}) + \langle\nabla\ell(\boldsymbol{\theta}), \boldsymbol{\vartheta} - \boldsymbol{\theta}\rangle + \frac{\mu}{2}\|\boldsymbol{\vartheta} - \boldsymbol{\theta}\|^2 \quad (\forall\, \boldsymbol{\vartheta},\ \boldsymbol{\theta}).$$

Note that $\mu = 0$ reduces to the special case of convex function. Also note that a positive definite quadratic function is strongly convex with $\mu = \lambda_{\min}$. For twice differentiable functions, we can summarize succinctly:

Proposition 4.3.3 Let ℓ be twice differentiable, μ-strongly convex and L-smooth, then

$$\mathbf{0} \prec \mu\mathbf{I} \preceq \nabla^2\ell(\boldsymbol{\theta}) \preceq L\mathbf{I}, \quad (\forall\boldsymbol{\theta})$$

→ Strong convexity lower bounds the smallest and the largest eigenvalue of a locally quadratic approximation of ℓ.

One may suspect that strong-convexity also avoids the vanishing gradient problem. Indeed, the PL condition is implied by strong convexity.

Proposition 4.3.4 Let ℓ be μ-strongly convex, then it fullfills the PL condition with the same μ.

Proof. Minimize both sides of the strong convexity condition

$$\begin{aligned}\ell^* - \ell(\boldsymbol{\theta}) = \min_{\boldsymbol{\vartheta}} \ell(\boldsymbol{\vartheta}) - \ell(\boldsymbol{\theta}) &\geq \min_{\boldsymbol{\vartheta}} \left\{\langle\nabla\ell(\boldsymbol{\theta}), \boldsymbol{\vartheta} - \boldsymbol{\theta}\rangle + \frac{\mu}{2}\|\boldsymbol{\vartheta} - \boldsymbol{\theta}\|^2\right\} \\ &= \langle\nabla\ell(\boldsymbol{\theta}), -\frac{1}{\mu}\nabla\ell(\boldsymbol{\theta})\rangle + \frac{\mu}{2}\left\|\frac{1}{\mu}\nabla\ell(\boldsymbol{\theta})\right\|^2 = -\frac{1}{2\mu}\|\nabla\ell(\boldsymbol{\theta})\|^2\end{aligned}$$

as the minimizer is $\boldsymbol{\vartheta} = \boldsymbol{\theta} - (1/\mu)\nabla\ell(\boldsymbol{\theta})$. From this the PL-condition follows by multiplying both sides of the inequality with $-\mu$. ■

A typical situation where strong convexity emerges is the use of convex objectives ℓ that include a norm-regularizer $\frac{\mu}{2}\|\boldsymbol{\theta}\|^2$.

→ In Deep Neural Networks, the PL condition will typically not hold globally, but possibly over a domain around a local minimum. It then ensures fast local convergence to this critical point without making claims to its sub-optimality.

4.3.4 Saddle Points

Q What happens to gradient descent around saddle points?

In the non-convex case, the training objective of a DNN may contain saddle points. One is then interested in the expected slow-down of gradient descent in the neighborhood of saddles. Or put differently: one is interested in how long it takes to escape from the vicinity of a saddle point. One generally strategy is then to add noise [22] or to exploit the noise induced by stochastic gradient descent [7]. A formal treatment is beyond the scope of this lecture.

→ Noisy gradient descent is a valid strategy to avoid slow-down in the vicinity of saddle points.

4.3.5 Momentum and Acceleration

As we have seen, one fundamental challenge in gradient descent arises from vanishing gradients (i.e. gradient norms becoming small). Requiring conditions like the PL condition is not solving anything, it is merely articulating a sufficient condition for plain gradient descent to work well. The perhaps more interesting question is:

Q How can gradient descent be modified to avoid a slow down in regions of small gradient norms?

The basic idea is to compensate small gradients by smoothness. If the objective is very smooth, one can increase the effective step size, overcoming – to some extend – small gradient norms. A very popular technique for implicit step size adjustment is known as *momentum*. The name refers to the motivational idea from classical mechanics of introducing a particle mass and of thinking of gradient descent as a discretization of the dynamics of such a particle in the gradient vector field. The conceptually simplest incarnation is the *heavy ball* method, which evolves iterates according to

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \nabla \ell(\boldsymbol{\theta}^k) + \beta(\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}), \quad \beta \in (0;1)\,. \tag{heavy ball}$$

Compared to plain gradient descent, there is a β-weighted term that lets the particle extrapolate the change made in the previous update. It is not fully understood under which conditions the heavy ball method has accelerated convergence, but it certainly can accelerate the transient phase of gradient descent and it is known to obey local acceleration [34]. Here, let us just provide a naïve explanation of what happens, if the gradient is constant over many steps (smoothness)

$$\begin{aligned}
\boldsymbol{\theta}^1 - \boldsymbol{\theta}^0 &= -\eta \nabla \ell \\
\boldsymbol{\theta}^2 - \boldsymbol{\theta}^1 &= -\eta(1+\beta)\nabla \ell \\
\boldsymbol{\theta}^3 - \boldsymbol{\theta}^2 &= -\eta(1+\beta(1+\beta))\nabla \ell = -\eta(1+\beta+\beta^2)\nabla \ell \\
&\cdots
\end{aligned}$$

So that in the limit

$$\lim_{k\to\infty} (\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}) = -\eta \sum_{i=0}^{\infty} \beta^i \nabla \ell = -\left[\frac{\eta}{1-\beta}\right] \nabla \ell \tag{4.39}$$

→ By using large momentum, i.e. $\beta \to 1$, one can (in principle) boost the effective step size by an arbitrarily large factor.

Practically speaking, taking β too large will create oscillations and instabilities as $\nabla \ell$ will not be constant. It is then often a matter of problem-specific hyper-parameter tuning to select β, commonly in the range $[0.9; 0.95]$. Note that by our back-of-the-envelope calculation the choice $\beta = 0.9$ provides the promise of a $10\times$ acceleration.

There is a related, famous method known as Nesterov acceleration, which pursues the same idea, but evaluates the gradient at the extrapolated point.

$$\boldsymbol{\vartheta}^{k+1} = \boldsymbol{\theta}^k + \beta(\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}), \tag{4.40}$$

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\vartheta}^{k+1} - \eta \nabla \ell(\boldsymbol{\vartheta}^{k+1})\,. \tag{4.41}$$

This subtle change makes a huge difference in the know guarantees of convergence for convex and strongly-convex functions. Nesterov acceleration is known to be (up to constants) optimal in the convex case and accelerated in the strongly-convex case. The heavy-ball method, however, is conceptually simpler and seems to be robustly result in benefits for non-convex functions.

→ Theoretically better understood than Polyak's heavy-ball method is Nesterov's accelerated gradient descent, which evaluates the gradient at the extrapolated iterate.

4.3.6 Adaptivity

It turns out that it is often important to be able to adapt the learning rate per parameter or dimension. This can be motivated from settings, where features are sparse, but also in compositional models, where it may be hard (or impossible) to find a single learning rate that yields good progress and convergence across a complex and deep architecture.

Q How can the effective step size be adapted per dimension?

The key idea is to use the history of gradients at previous iterates to influence the effective step size. Let us define the monotonically increasing sequence

$$\gamma_i^k = \gamma_i^{k-1} + \left[\partial_i \ell(\boldsymbol{\theta}^k)\right]^2, \quad \partial_i \ell := \frac{\partial \ell}{\partial \theta_i} \tag{4.42}$$

γ_i^k corresponds to the sum of the squares of the i-th parameter's partial derivatives along the iterate sequence $\boldsymbol{\theta}^0, \dots, \boldsymbol{\theta}^k$. We can use these estimates to adapt the step size per parameter, i.e. by using a diagonal – so-called – pre-conditioner

$$\theta_i^{k+1} = \theta_i^k - \eta_i^k \partial_i \ell(\boldsymbol{\theta}^k), \quad \eta_i^k := \frac{\eta}{\sqrt{\gamma_i^k} + \delta} \tag{AdaGrad}$$

with $\delta > 0$ (small, for numeric stability). This resulting variant of gradient descent is called *AdaGrad*. Parameters with historically smaller magnitudes of their partial derivatives are updated with an effectively larger step size.

→ Adaptivity can be obtained by making use of the entire gradient history across all iterates.

There is a sophisticated analysis of AdaGrad using regret bounds for the case of convex objectives, see [11]. The details of this analysis are beyond our current scope. Practically speaking, Adagrad drives the learning rate eventually to zero.

4.3.7 Adam and RMSprop

Q How can stochasticity, adaptivity and momentum be combined in a way that works well for non-convex objectives? Or put differently: *What is the state-of-the-art learning algorithm for DNNs?*

We now come to discuss what can be rightfully called the state of the art learning algorithm in DNNs, known as *Adam* (Adaptive Momentum Estimation). Adam uses an exponentially weighted average to estimate the mean and variance of each partial derivative

$$g_i^k = \beta g_i^{k-1} + (1-\beta)\partial_i \ell(\boldsymbol{\theta}^k), \quad \beta \in [0;1], \quad g_i^0 := \partial_i \ell(\boldsymbol{\theta}^0) \tag{4.43}$$

$$h_i^k = \alpha h_i^{k-1} + (1-\alpha)[\partial_i \ell(\boldsymbol{\theta}^k)]^2, \quad \alpha \in [0;1], \quad h_i^0 := [\partial_i \ell(\boldsymbol{\theta}^0)]^2 \tag{4.44}$$

and then defines the iterate sequence

$$\theta_i^{k+1} = \theta_i^k - \eta_i^k g_i^k, \quad \eta_i^k := \frac{\eta}{\sqrt{h_i^k} + \delta}. \tag{4.45}$$

Note that the specific form of the update leads to an invariance with regard to re-scaling of the partial derivatives, i.e. if $\partial_i \ell$ is rescaled by some ρ_i, this will cancel out (assuming $\delta \to 0$) as g_i^k and $\sqrt{h_i^k}$ will be re-scaled by ρ_i. In the case of axis-aligned quadratics, the allows to adapt to the curvature. More details on Adam can be found in [24]. Adam without the use of momentum (i.e. $\beta = 0$) is also known as RMSprop.

→ Practically successful is the use of exponential averaging of partial derivatives and their variance to implement momentum and adaptivity in schemes like Adam.

Of course, the gradient used in Adam will typically be stochastic based on some minibatch of data points.

4.4 Convolutional Neural Networks

Much of the success of DNNs in the context of machine perception, in particular in machine vision, has exploited convolutional layers and convolutional neural networks (CNNs). These networks exploit certain invariances of the input patterns such as translational invariance (and equivariance). Another way to look at this is that CNNs extract (the same localized) features at every spatial or temporal location. We will start by introducing convolutions.

4.4.1 Convolutions

Integral Operators

Let us assume – for concreteness - we have a time-signal f, i.e. a function $f : \mathbb{R} \to \mathbb{R}^m$. We are interested in transforming f into a function $Tf : \mathbb{R} \to \mathbb{R}^m$ via an operator T. The motivation is to extend the finite dimensional case of mapping $\mathbf{x} \in \mathbb{R}^n$ to a feature representation $\mathbf{x} \mapsto \phi(\mathbf{x}) \in \mathbb{R}^m$, e.g. via a shallow learning machine or via a processing layer in a DNN.

Q How can we represent an interesting class of operators for transforming functions?

An important class of operators are *integral operators*, which rely on a representation via kernels.

Definition 4.4.1 — Integral Operator. Given a kernel $H : \mathbb{R}^2 \to \mathbb{R}$ and an interval $t_1, t_2, \in \mathbb{R} \cup \{-\infty, \infty\}$ we can define an operator (assuming the integral exists) via

$$(Tf)(u) = \int_{t_1}^{t_2} H(u,t) f(t)\, dt$$

Proposition 4.4.1 Integral operators are linear.

Proof. Follows directly from the linearity of the integral. ∎

In fact, if one extends the class of functions defining integral operators to generalized functions known as *distributions*, then (essentially) every linear operator can be represented as an integral operator. This is formally described by the Schwartz kernel theorem.

→ Essentially all linear operators have an associated kernel representation as an integral operator.

Continuous Convolution

A particularly relevant class of integral operators are convolutions.

Definition 4.4.2 — Convolution. Given two functions f, h, their convolution is defined as

$$(f * h)(u) := \int_{-\infty}^{\infty} h(u-t) f(t)\, dt = \int_{-\infty}^{\infty} f(u-t) h(t)\, dt$$

This corresponds to an integral operator with kernel $H(u,t) = h(u-t)$ operating on f. Or, by symmetry, we can think of $F(u,t) = f(u-t)$ operating on h.

Q What is special about convolutions and the operators they define?

Let us define a shifted function f_Δ

$$f_\Delta(t) := f(t+\Delta) \tag{4.46}$$

and consider the natural notion of translational or shift invariance.

Definition 4.4.3 — Shift Invariant Operator. An operator T is shift invariant, if for any $\Delta \in \mathbb{R}$ and function f in its domain

$$T(f_\Delta) = (Tf)_\Delta$$

Proposition 4.4.2 Let h be an arbitrary kernel. Then, the integral operator T defined by $Tf := f * h$ defines a shift-invariant operator.

Proof.

$$\begin{aligned}(f_\Delta * h)(u) &= \int_{-\infty}^{\infty} h(u-t) f(t+\Delta)\, dt \\ &\overset{t \mapsto t-\Delta}{=} \int_{-\infty}^{\infty} h(u+\Delta-t) f(t)\, dt = (f*h)(u+\Delta) = (f*h)_\Delta(u)\end{aligned}$$

∎

The essential property exploited is the shift invariance of the kernel $H(u-\Delta, t-\Delta) = h(u-t) = H(u,t)$ $(\forall \Delta)$. Some additional comments: (1) The convolution operator is commutative, one can exchange the function and the kernels. (2) Its existence depends on integrability properties of f, h. (3) Typical use: f = signal, h = fast decaying kernel function, often of finite support.

→ Convolutions define shift-invariant linear operators.

In fact, the converse is also true.

→ Any linear, translation-invariant transformation T can be written as a convolution with a suitable kernel h.

Discrete Convolution

In practice signals are typically (digitally) sampled and we need to consider the discrete case. Let $f, h : \mathbb{Z} \to \mathbb{R}$. We can define the discrete convolution via

$$(f * h)[u] := \sum_{t=-\infty}^{\infty} f[t]\, h[u-t] \qquad \text{(discrete 1d convolution)}$$

We use rectangular brackets to suggest "arrays". It is easy to extend this to two dimensions:

$$(f * h)[x,y] := \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u,v]\, h[x-u, y-v] \qquad \text{(discrete 2d convolution)}$$

A typical choice of h may involve a finite support in the form of a window size. For instance $h[t] = 0$ for $t \notin [t_{\min}; t_{\max}]$. In practice, f will often also be defined only over a finite domain.

Example 4.1 Let us define a small Gaussian kernel with support $[-2:2] \subset \mathbb{Z}$.

$$h[t] = \frac{1}{16} \begin{cases} 6 & t = 0 \\ 4 & |t| = 1 \\ 1 & |t| = 2 \\ 0 & \text{otherwise} \end{cases}$$

As a consequence of the finite support, the convolution sum can be truncated

$$\begin{aligned} (f * h)[u] &= \sum_{t=u-2}^{u+2} f[t]\, h[u-t] = \sum_{t=-2}^{2} h[t]\, f[u-t] \\ &= \frac{6f[u] + 4f[u-1] + 4f[u+1] + f[u-2] + f[u+2]}{16} \end{aligned}$$

■

There is a sibling to convolutions, known as cross–correlation. In the discrete case let $f, h : \mathbb{Z} \to \mathbb{R}$, then

$$(h \star f)[u] := \sum_{t=-\infty}^{\infty} h[t]\, f[u+t] \qquad \text{(cross–correlation)}$$

This is also called a “sliding inner product", the difference is just one of sign: $u + t$ instead of $u - t$. This can be made even clearer by writing

$$(h \star f) = (\bar{h} * f), \quad \text{where } \bar{h}[t] := h[-t], \tag{4.47}$$

which follows from

$$(h \star f)[u] = \sum_{t=-\infty}^{\infty} h[t]\, f[u+t] = \sum_{t=-\infty}^{\infty} h[-t]\, f[u-t] = (\bar{h} * f`)[u] \tag{4.48}$$

So the only difference is that the kernel is “flipped over". This has the side-effect to make cross-correlations non-commutative.

4.4.2 Convolutional Networks

Goals

Q What is our goal with convolutional neural networks?

The idea of CNNs is to learn convolutional kernels and thereby transformations of discretized signals, often over 1 (e.g. speech), 2 (e.g. images) or 3 (e.g. videos) dimensional grids. If one restricts the support of the kernel, one can easily parameterize it finitely. Typically, one does not implement convolutions, but – in an abuse of terminology – cross-correlations. The following goals are associated with a CNN

- a compositional (i.e. layered) architecture for signal processing
- exploiting *shift invariance*
- exploiting locality and scale (e.g. temporal, spatial)
- learning (parameterized) kernel functions or *filters*
- increased statistical efficiency (vs. fully-connected DNNs)

→ Learn shift invariant linear transformations of signals by learning parameters of convolutional kernels (or filters).

Computer Vision

Computer vision is an area where convolutional networks have been used with remarkable success. Note that a pixel-based image representation is usually 3d, a regular two dimensional sampling grid on which 3 color channels are measured (dependent on color models such as RGB). Convolutional kernels are thus usually operating on 3-*tensors*. However the channel dimension is permutation invariant and their numbering arbitrary.

The layer map of a CNN (with ReLU) can be formally written as follows:

$$\underbrace{z_{i,(u,v)}}_{\text{3-tensor}} = \left(\sum_{j=1}^{n} \sum_{(\delta u, \delta v) \in \Delta} \underbrace{\theta_{i,j,(\delta u, \delta v)}}_{\text{4-tensor}} \underbrace{x_{j,(u+\delta u, v+\delta v)}}_{\text{3-tensor}} \right)_+ , \quad i = 1, \dots, k \tag{4.49}$$

Here, j sums over the n input channels, i indexes k output channels, and the range of summation over offsets $\delta u, \delta v$ is determined by the window Δ. The number of parameters

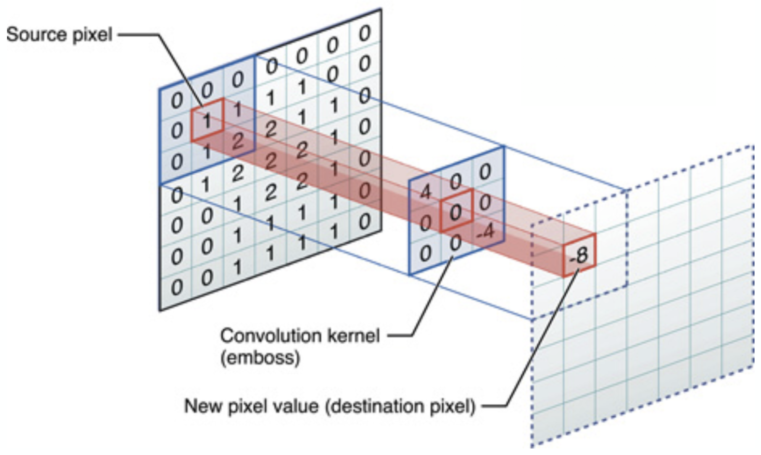


Figure 4.4: 2d Convolution: single channel

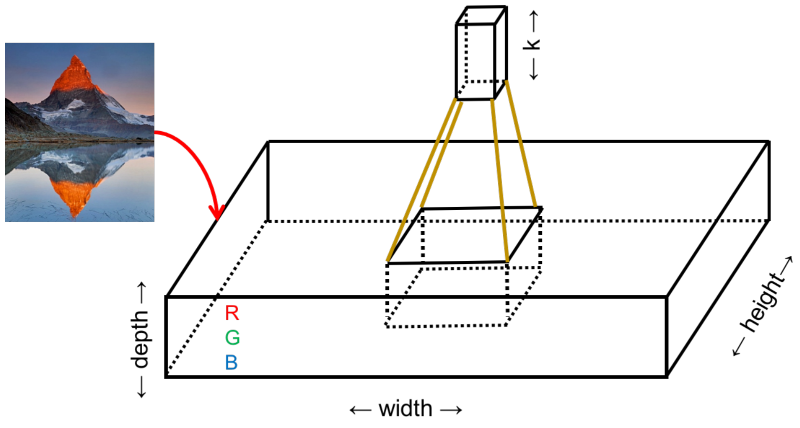


Figure 4.5: Diagram showing a convolutional kernel for color images.

scales with $n \cdot k \cdot |\Delta|$. A single channel view of a 2d-convolution is shown in Figure 4.4, whereas a multichannel convolution on an RGB input is depicted in Figure 4.5.

In addition to the discrete convolution and the pointwise non-linearity, there are commonly used additional processing steps such as down-sampling (also known as strides) of feature maps in order to reduce the dimensionality of the model. Down-sampling is often combined with an operation called *pooling*, which can also be applied on its own. A common parameter-free pooling layer is *max-pooling* over a window Δ defined as follows

$$z_{i,(u,v)} = \max_{(\delta u, \delta v) \in \Delta} x_{i,(u+\delta u, v+\delta v)}$$

Q What does a suitable CNN architecture for computer vision look like?

The classical CNN architecture is perhaps best illustrated with a famous network known as AlexNet [28] (current citation count 80k+) shown in Figure 4.6. One can see the downsampling by strides of 2 (with a border effect) and an increase from channels from 3 (RGB) to 384. The transition from the convolutional layer to the first fully-connected layer is critical in terms of model dimensionality.

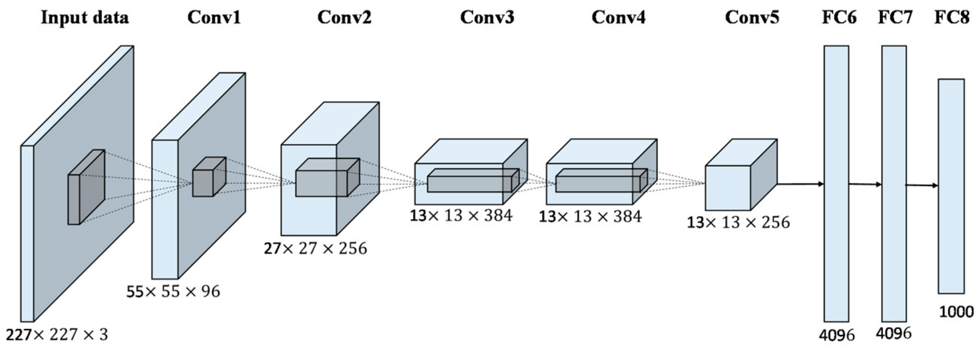


Figure 4.6: AlexNet from [28]

→ Typical CNN architecture for vision: pyramidal structure. With depth, less resolution, more channels, fully-connected in the end.

One major innovation in CNN architectures for computer vision has been the use of residual networks (ResNets, [19], 77k+ citations), which introduce shortcut connections to avoid the vanishing gradient problem (see Figure 4.7).

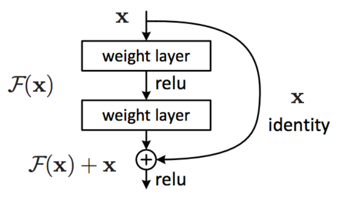


Figure 4.7: Residual layer: block

ResNets allow for training of very deep models with hundreds of layers (see Figure 4.8).

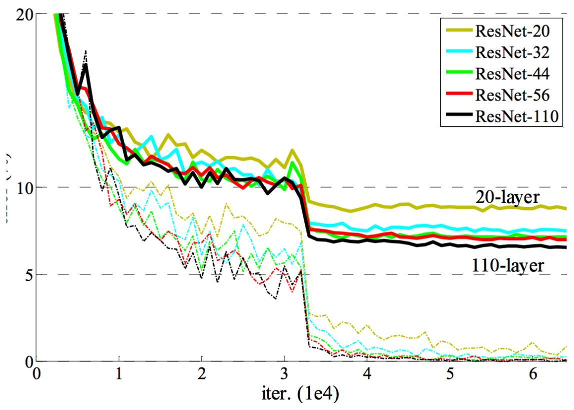


Figure 4.8: ImageNet classification accuracy obtained by ResNets of different depth.

→ CNNs and ResNets have been major drivers of the perceptual AI revolution.

5. Generative Models

Let us consider the following setting. Patterns $\mathbf{x}$ are generated from some process with law $p(\mathbf{x})$. We define a $\boldsymbol{\theta}$-parameterized model $p(\mathbf{x};\boldsymbol{\theta})$ and want to chose $\boldsymbol{\theta}$ such hat $p(\mathbf{x})$ and $p(\mathbf{x};\boldsymbol{\theta})$ become indistinguishable. Specifically, we are interested in models that do not generate unnatural patterns and that generate natural patterns with approximately correct probabilities. That is we want to avoid

$$p(\mathbf{x}) \gg p(\mathbf{x};\boldsymbol{\theta}) \gtrapprox 0 \qquad \text{(mode collapse)}$$

$$p(\mathbf{x};\boldsymbol{\theta}) \gg p(\mathbf{x}) \gtrapprox 0 \qquad \text{(incorrect patterns)}$$

For example, if the patterns are images of human faces, then we want to only generate images that look like faces and if there are particular distributions of facial poses or expressions, say, then we want the model to reproduce those in the correct proportions. For many data sources of interest like images, audio, or video, the final yardstick of success is the *perceptive indistinguishability* by humans. This motivation may sound straightforward (and perhaps naïve), but it constitutes a significant departure from the type of models and objectives traditionally investigated and deployed in statistics for many decades.

Q How can we formalize the notion of generative models and the ultimate criterion of success as a machine learning problem?

5.1 Autoregressive Models

5.1.1 Likelihood-Based Model Inference

Let us start with the classical view in statistics. Learning a generative model may be considered equivalent to the problem known as *density estimation.* The canonical inference principle will be MLE (maximum likelihood estimation), which means we will chose $\boldsymbol{\theta}^*$ such that

$$\boldsymbol{\theta}^* \overset{\max}{\longrightarrow} \mathbf{E}[\,\ln p(\mathbf{x};\boldsymbol{\theta})\,], \quad \mathbf{x} \sim p(\mathbf{x})\,. \tag{5.1}$$

In practice, the expectation will typically be with regard to an empirical sample (aka training set). Traditionally, model families under considerations have often been simple, for instance belonging to exponential (sub-)families or mixture models, where a specific parametric assumption is made.

R We have discussed mixture distributions as an example, where the main difficulty was the non-concavity of the log-likelihood, which was (heuristically) overcome by the use of Expectation-Maximization algorithm.

We will here focus on a specific problem of density estimation, namely one where the data is high-dimensional as is common in AI when working with raw sensor data as images, audio, or videos. Yet note that as we discussed in the context of convolutional neural networks (CNNs), such data often obeys shift-invariance, which – when used intelligently – counteracts the curse of dimensionality.

Q How can we learn probability densities of high-dimensional, shift-invariant data?

5.1.2 Autoregressive Models

The conceptually simplest approach is to define models directly on the observables without having to deal with latent representations (hidden layers) in DNNs. One fruitful idea is the use of the chain rule to make models *autoregressive*. We assume that variables are sequentially ordered and write

$$p(\mathbf{x};\boldsymbol{\theta}) = p(x_1,\dots,x_m;\boldsymbol{\theta}) = \prod_{t=1}^{m} p(x_t|x_1,\dots,x_{t-1};\boldsymbol{\theta})\,. \tag{5.2}$$

→ Autoregressive models break down the problem of generating high-dimensional patterns by conditionally generating one variable at a time.

Note that often the conditioning context may be reduced, for instance by making a k-th order Markov assumption. In the case of sequential data

$$p(x_t|x_1,\dots,x_{t-1};\boldsymbol{\theta}) \overset{\text{Markov}}{=} p(x_t|x_{t-k},\dots,x_{t-1};\boldsymbol{\theta}) \qquad (k\text{-th order Markov chain})$$

Markov assumptions further simplify the model and unify the prediction problem, which now always operates with a fixed size context.

R In areas like speech recognition and natural language processing, autoregressive models with discrete variables have been used to learn language models based on so-called word n-grams.

The classical linear model is the AR(k) model defined with white Gaussian noise as

$$X_t = \theta_0 + \sum_{i=1}^{k} \theta_i X_{t-i} + \epsilon_t, \quad \epsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0,\sigma^2) \tag{5.3}$$

Such models and related ones like ARMA (autregressive-moving-average model) are very common in time-series prediction. Note that autoregressive models operate strictly sequentially. In the context of generative models, influential models that have been used with some success include PixelCNN for image generation [40] and WaveNet for speech synthesis [32].

5.1.3 PixelCNN

We will focus on PixelCNNs and define some scanning order for pixel (e.g. top-to-bottom and left-to-right). The network then models the conditional distribution of an individual pixel given preceding pixels. During image generation, one can generate the image sequentially pixel-by-pixel in the chosen order (difficult to parallelize). One can think of the generation process in terms of a masked convolution, where pixels that have not yet been generated are masked out in the convolutional kernel. A simple example of a masked is shown in Figure 5.1. The conditional model is discrete, i.e. in the case of grayscale images, each of

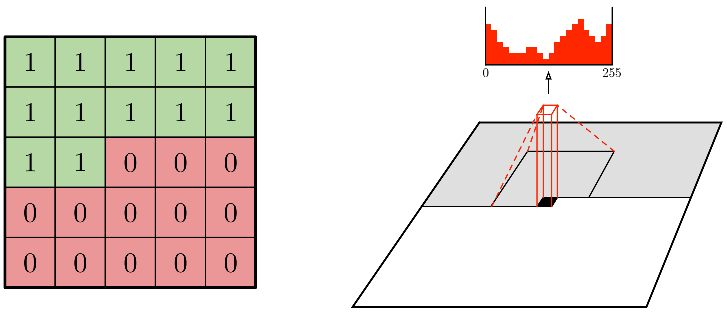


Figure 5.1: Masked 5×5 convolutional kernel and a sketch of the image scanning process.

the 256 possible values has a probability (i.e. a histogram). In the case of color images, RGB channels are generated sequentially. Some example images generated with PixelCNN are shown in Figure 5.2. =

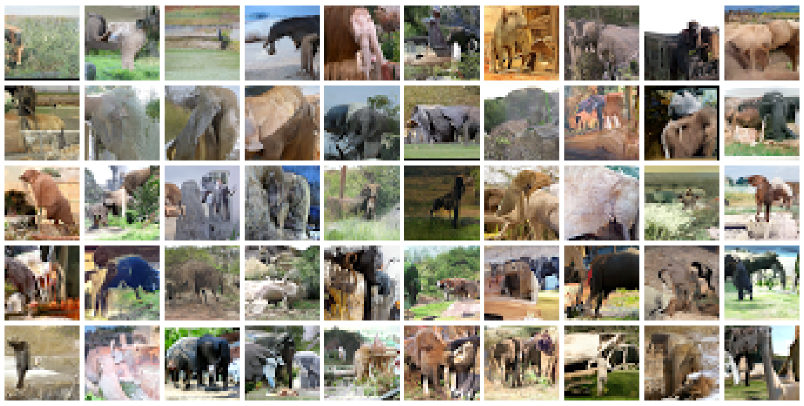


Figure 5.2: Synthetic images of African elephants (from [40])

5.2 Normalizing Flows

5.2.1 Implicit Models

Q Can we leverage the power and flexibility of Deep Neural Networks for generative models?

Simple parametric models are often too limited in their representational power to be useful as generative models for more challenging, high-dimensional problem.[1] It has turned out, however, that it is possible to leverage the power of DNNs for generative models. The basic idea relates to latent variable models. Assume that we sample a latent code $\mathbf{z}$ from some simple (often fixed) distribution such as a normal distribution. One then transforms $\mathbf{z}$ into a pattern $\mathbf{x}$ with the help of a DNN, often with a purely deterministic map. This can be thought of a transformation of random variables

$$X = F(Z;\boldsymbol{\theta}), \quad Z \sim \mathcal{N}(0,\mathbf{I})\,. \tag{5.4}$$

This *implicit* model transforms the original (simple) distribution, but does not explicitly prescribe its density. A diagrammatic view of this approach to generative modeling is shown in Figure 5.3 What makes this approach attractive is the so-called *Law of the Unconscious*

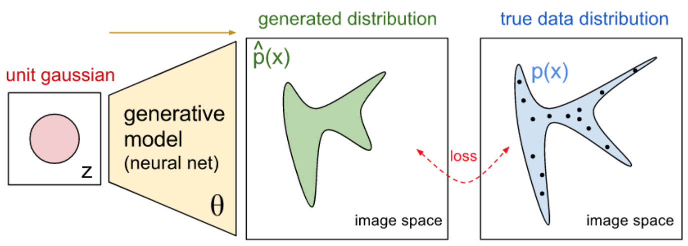


Figure 5.3: An implicit model uses a DNN to transform a simple low-dimensional distribution in code space into a complicated high dimensional distribution in pattern space.

Statistician, which can be mathematically expressed as follows:

$$\mathbf{E}_X[g(X)] = \int (g \circ F)(\mathbf{z})\, p_Z(\mathbf{z}) d\mathbf{z}\,. \tag{5.5}$$

Intuitively (and practically) it states that one can can compute or approximate expectations with regard to a transformed random variable $\mathbf{x} = F(\mathbf{z})$ by sampling $\mathbf{z}$ and pushing samples through the mapping F to obtain samples of $\mathbf{x}$. As we will see below, Monte Carlo approximations of expectations are often acceptable and sufficient. By lowering our requirements and moving away from analytic representations, we will be able increase our modeling power substantially.

→ Transforming random variables with DNNs leads to powerful implicit models one can easily sample from.

In comparison to explicit or prescribed models we summarize:

[1] Traditional non-parametric methods also typically fail for high-dimensional problems.

→ While a prescribed model specifies the density explicitly, an implicit model specifies the transformation of samples instead.

While the above property is pivotal from a practical perspective, we may ask:

Q What is the density of an implicit model that transforms a given random variables ?

To answer this question, we have to consider a formula known as integration by substitution. We will state a version known as the bi-Lipschitz version.

Proposition 5.2.1 Let $U \subseteq \mathbb{R}^n$ be open and $F : \mathbb{R}^n \to \mathbb{R}^m$ be bi-Lipschitz map (Lipschitz with a Lipschitz inverse). For any measurable $h : F(U) \to \mathbb{R}$

$$\int_U (h \circ F)(\mathbf{z}) \det|\mathbf{J}_F(\mathbf{z})| d\mathbf{z} = \int_{F(U)} h(\mathbf{x})\, d\mathbf{x}$$

where $\mathbf{J}_F$ is the Jacobi matrix of F.

From this we obtain the following rule for the transformation of probability densities

$$\mathbf{x} = F(\mathbf{z}), \quad p_X(\mathbf{x}) = p_Z(F^{-1}(\mathbf{x}))\, |\det(\mathbf{J}_F)|^{-1} \tag{5.6}$$

p_X is called the *pushforward* of p_Z.

→ Implicit models define probability densities on observables via the push-forward of the latent code density. Evaluating the pushforward involves the inverse map and its Jacobian determinant.

Q How can we restrict Deep Neural Networks to be able to perform Maximum Likelihood Estimation?

5.2.2 Compositionality

The idea of *normalizing flows* is to work with bijections F which are convenient to compute, invert, and for which we can efficiently calculate $|\det(\partial F)|$. We first note that for a compositional map, it is sufficient to enforce these properties for the layer maps.

$$F = F_L \circ \cdots \circ F_1, \quad F^{-1} = F_1^{-1} \circ \cdots \circ F_L^{-1} \tag{5.7}$$

$$\det(\mathbf{J}_F) = \prod_{l=L}^{1} \det(\mathbf{J}_{F_l} \circ F_{l-1:1}), \quad \det(\mathbf{J}_{F^{-1}}) = \det(\mathbf{J}_F)^{-1} \tag{5.8}$$

We can then write the log-likelihood

$$\ln p_X(\mathbf{x}) = \ln p_Z(F^{-1}(\mathbf{x})) - \sum_{l=1}^{L} \ln |\det (\mathbf{J}_{F_l} \circ F_{l-1:1})\,| \tag{5.9}$$

5.2.3 Affine Flows

Let us first study the simplest class of normalizing flows, those defined by affine transformations with an invertible linear map $\mathbf{A} \in \mathbb{R}^{n\times n}$. We get

$$\mathbf{x} = F(\mathbf{z}) = \mathbf{A}\mathbf{z} + \boldsymbol{\mu}, \quad \mathbf{J}_F = \mathbf{A}, \quad \mathbf{z} = F^{-1}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} - \boldsymbol{\mu}), \quad \mathbf{J}_{F^{-1}} = \mathbf{A}^{-1}. \tag{5.10}$$

If $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then we get $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{A}\mathbf{A}^\top)$. So affine flows transforms an iid standard normal random vector into a multivariate normal random vector with mean given by $\boldsymbol{\mu}$ and variance-covariance matrix $\mathbf{A}\mathbf{A}^\top$. Note that composing multiple such flows does not increase the modeling power.

5.2.4 Planar and Sylvester Flows

Q How can we define a simple enough flow that is compositional and has an easy to compute Jacobian determinant?

Let us consider a simple non-linear flow that effectively uses a layer with a single non-linear unit with activation function ϕ to additively modify the identity function (cf. [36])

$$F(\mathbf{z}) = \mathbf{z} + \phi(\langle \boldsymbol{\theta}, \mathbf{z} \rangle + b)\mathbf{u}. \tag{5.11}$$

This model has $2n+1$ parameters and changes the distribution in the direction of $\mathbf{u}$, i.e. one dimension at a time. Clearly, in order to transform all n dimensions, at least depth n is needed. In general, as planar flows only allow for very limited layer maps, substantial depth may be needed and the approach may only work well in small to medium dimensions. The advantage of this formulation is that the the determinant can be explicitly calculated

$$|\det(\mathbf{J}_F)| = |\det(\mathbf{I} + \phi'(\langle \boldsymbol{\theta}, \mathbf{z} \rangle + b)\mathbf{u}\boldsymbol{\theta}^\top)| = |1 + \phi'(\langle \boldsymbol{\theta}, \mathbf{z} \rangle + b)\,\langle\, \mathbf{u}, \boldsymbol{\theta} \rangle| \tag{5.12}$$

The last equality follows from the matrix determinant lemma by setting $\mathbf{A} = \mathbf{I}$.

Lemma 5.2.2 — Matrix Determinant Lemma.

$$\det(\mathbf{A} + \mathbf{u}\mathbf{v}^\top) = (1 + \mathbf{v}^\top \mathbf{A}^{-1}\mathbf{u}) \det(\mathbf{A})$$

Corollary 5.2.3

$$\det(\mathbf{I} + \mathbf{u}\mathbf{v}^\top) = (1 + \mathbf{u}^\top \mathbf{v})$$

Some examples of planar flows with different depth K are shown in Figure 5.4. One can generalize this idea by allowing for $m \le n$ units in the bottleneck layer and embedding this m-dimensional subspace in $\mathbb{R}^n$ via $\mathbf{U} \in \mathbb{R}^{n\times m}$

$$F(\mathbf{z}) = \mathbf{z} + \mathbf{U}\phi(\mathbf{A}\mathbf{z} + \mathbf{b}) \tag{5.13}$$

This model has also been called Sylvester flow as it makes use of the Sylvester identity for determinants.

Lemma 5.2.4 — Sylvester's Determinant Identity. Let $\mathbf{U}, \mathbf{V}^\top \in \mathbb{R}^{n\times m}$, then

$$\det(\mathbf{I}_n + \mathbf{U}\mathbf{V}) = \det(\mathbf{I}_m + \mathbf{V}\mathbf{U})$$

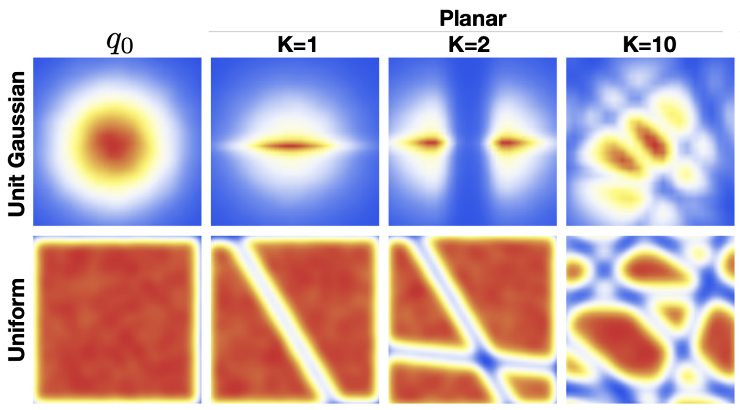


Figure 5.4: Depth K planar flow transformations of a Gaussian and the uniform density (figure taken from [36]).

By comparison to planar normalized flows, there is more flexibility in trading-off the complexity of the determinant of a $m \times m$ matrix vs. the increased modeling power.

→ Planar and Sylvester flows are simple instances of compositional normalizing flows for problems of medium dimensionality (10s to 100s).

5.2.5 Invertible 1×1 Convolutions

Q What other possibilities exist to define normalizing flows?

There is a wealth of models that has been suggested to control the trade-off of modeling power vs. the extra cost of dealing with the Jacobian determinant. We here select a model known as *Glow* [25], which has shown promise in the context of generating high dimensional data like images. In the case of images, the model operates on 3-tensors with spatial dimension $p \times q$ and r channels. The model transforms tensors through three layers: (1) a normalization layer, (2) a 1×1 convolution and (3) an affine coupling layer.

The normalization layer allows for an explicit control of the mean and variance of the per channel response.[2] The normalization and the convolution operate on the channel dimension and one has independently at each spatial location (i, j):

$$\mathbf{z}_{ij}^{+} = \mathbf{W}\left(\text{diag}(\mathbf{s})\mathbf{z}_{ij} + \mathbf{b}\right), \quad 1 \le i \le p,\ 1 \le j \le r \tag{5.14}$$

It turns out that such a mapping can be parameterized via the LU decomposition as

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \text{diag}(\mathbf{s})) \tag{5.15}$$

Where $\mathbf{L}$ is lower triagonal with ones on the diagonal and $\mathbf{U}$ is upper diagonal with zero diagonal. $\mathbf{P}$ is a permutation matrix that can be chosen fixed (at random). It follows that the determinant only depends on $\mathbf{s}$ as

$$|\det(\mathbf{W})| = |\det(\text{diag}(\mathbf{s}))| \tag{5.16}$$

which relies on the following simple lemma.

[2] A detailed discussion requires detailed background knowledge about batch normalization and related techniques, which is beyond the scope of this lecture.

Lemma 5.2.5 Let $\mathbf{P}$ be a permutation matrix, $\mathbf{L}, \mathbf{U}$ be lower and upper triagonal, respectively. Then:

$$|\det(\mathbf{PLU})| = |\det(\operatorname{diag}(\mathbf{L}))\,\det(\operatorname{diag}(\mathbf{U}))| = \left|\left(\prod_i l_{ii}\right)\left(\prod_i u_{ii}\right)\right|$$

The affine coupling layer as suggested in [9, 10] is applied as a split along the channel dimension and passes through one half of the variables, while the other half is modulated by a function of the first. Define index sets

$$\alpha, \beta \subset \{1, \ldots, n\}, \quad \alpha \cup \beta = \{1, \ldots, n\}, \quad \alpha \cap \beta = \emptyset, \tag{5.17}$$

then with NN denoting a neural network mapping

$$(\mathbf{z}_\alpha^{++}, \mathbf{z}_\beta^{+}), \quad \mathbf{z}_\alpha^{++} = \operatorname{diag}(\boldsymbol{\sigma})\,\mathbf{z}_\alpha^{+} + \boldsymbol{\tau}, \quad (\boldsymbol{\sigma}, \boldsymbol{\tau}) = \mathrm{NN}(\mathbf{z}_\beta^{+}) \tag{5.18}$$

This form ensures again that the log determinant just depends on $\boldsymbol{\sigma}$. It is applied iteratively by performing the random partition into index sets α and β at random.

To illustrate the effectiveness of Glow-based generation, Figure 5.5 shows examples of synthetically generated images.

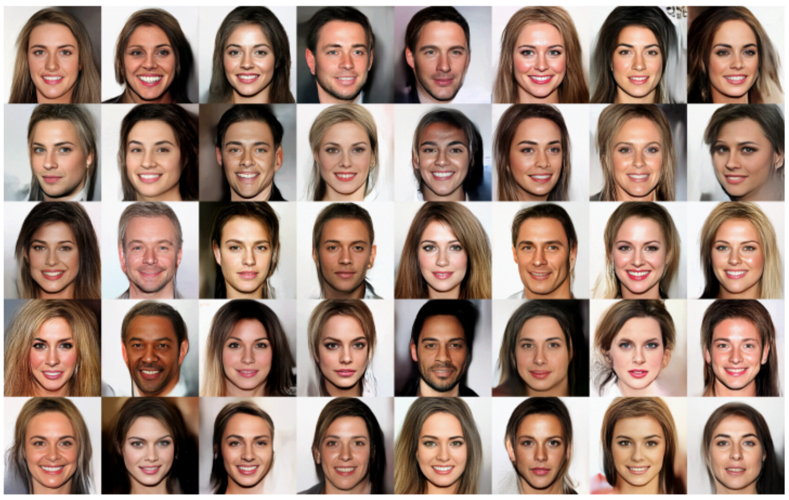


Figure 5.5: Faces generated by Glow, taken from [25]

→ There are many advanced normalized flow modules and architectures (r.g. Glow) that result in very promising generative models.

5.3 Variational Autoencoders

Normalizing flows aim to retain a tractable likelihood function by constraining the architecture to be invertible and the Jacobian determinant to be efficiently accessible. Another strategy is to approximate the likelihood function by some lower bound and to maximize the bound. We have seen this variational strategy with a parameterized family of lower bounds be utilized with success for mixture models and topic models. We will now further elaborate this idea in an approach known as variational autoencoders (VAEs).

5.3.1 Evidence Lower Bound

We start again with the Evidence Lower BOund (ELBO).

$$\log p(\mathbf{x};\boldsymbol{\theta}) = \log \int p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})\, p(\mathbf{z})d\mathbf{z} = \log \int q(\mathbf{z}) \left[p(\mathbf{x}|\mathbf{z};\theta) \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] d\mathbf{z} \tag{5.19}$$

$$\geq \int q(\mathbf{z}) \log p(\mathbf{x}|\mathbf{z};\theta) d\mathbf{z} - \underbrace{\int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})} d\mathbf{z}}_{=D_{\mathrm{KL}}(q\,||\,p)} =: \ell(\boldsymbol{\theta}, q; \mathbf{x}) \tag{5.20}$$

via Jensen's inequality for any choice of the variational distribution over encodings q. Now the basic idea in VAEs is to learn the variational parameters: instead of chosing a q for every $\mathbf{x}$, we learn a parameterized model and get the VAE objective

$$\ell(\boldsymbol{\theta}, \boldsymbol{\psi}; \mathbf{x}) = \int q(\mathbf{z}|\mathbf{x};\boldsymbol{\psi}) \log p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta}) d\mathbf{z} - D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x};\boldsymbol{\psi})\,||\,p(\mathbf{z})) \tag{5.21}$$

which depends on the parameters $\boldsymbol{\theta}$ of the generative model as well as the parameters $\boldsymbol{\psi}$ of the so-called *inference model.*

→ The variational ELBO can be implemented as a separate model known as the inference model.

5.3.2 Inference Models

Q How can we design suitable inference models?

Inference models are probabilistic models that generate a latent code $\mathbf{z}$ from a pattern $\mathbf{x}$. One may wonder: What is gained by having to learn two probabilistic models instead of just one? And in which ways are the requirements on inference models different from those on generative models? There is a general belief that a conditional distribution that conditions on a high-dimensional pattern to produce a lower dimensional code needs to be less powerful than a model that transforms a simple code distribution into a high-dimensional data law. A common choice is therefore one, where the stochasticity in the inference model is only added-in at the end. For instance a typical approach is to learn a deterministic mapping and to add noise via a learned, anisotropic covariance matrix, i.e.

$$\mathbf{E}[\mathbf{z}|\mathbf{x};\boldsymbol{\psi}] = \underbrace{\boldsymbol{\mu}}_{\text{mean fct}} \circ \underbrace{G(\mathbf{x};\boldsymbol{\psi})}_{\text{DNN}}, \quad \mathrm{Cov}[\mathbf{z}] = \underbrace{\boldsymbol{\Sigma}}_{\text{cov fct}} \circ \underbrace{G(\mathbf{x};\boldsymbol{\psi})}_{\text{DNN}} . \tag{5.22}$$

where it is assumed that $q(\mathbf{z}|\mathbf{x})$ is a normal distribution. The DNN produces a penultimate representation from which an output layer predicts the parameters of the normal distribution. Often the covariance model is further constrained, for instance to be a diagonal matrix plus a low-rank matrix. In the case of an unconstrained covariance matrix, we already know that we can model a multivariate normal as a linear flow of iid standard normals. So we can alternatively define

$$\mathbf{z} = \boldsymbol{\mu}(G(\mathbf{x})) + \mathbf{A}(G(\mathbf{x}))\, \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.23}$$

Sampling $\mathbf{z}\,|\,\mathbf{x}$ then involves (i) deterministically computing $G(\mathbf{x})$ and then (ii) randomly sampling one or more iid instantiations of $\boldsymbol{\epsilon}$. One can also extend the inference model to compose the dimension reduction map G by a normalizing flow (other than the linear one). In fact this is an area, where normalizing flow have been first proposed and investigated. In this more general case the DNN will predict the parameters of the flow as a function of $\mathbf{x}$.

→ Normalizing flows are suitable transformations that can be combined with DNNs to implement inference models.

5.3.3 VAE Learning

Note that the conditional model $p(\mathbf{x}|\mathbf{z})$ is deterministic, possibly feeding into a squared error loss, i.e. Gaussian log-likelihood, between a DNN-predicted $\hat{\mathbf{x}}$ and a training pattern $\mathbf{x}$. It is then a standard approach to perform a Monte Carlo approximation of the expectation w.r.t $q(\mathbf{z}|\mathbf{x})$. We can simply sample from q and use those samples as inputs to the generative model. We can then train the latter via standard backpropagation.

$$\mathbf{x} \begin{array}{c}\overset{\text{infer}}{\mapsto}\\ \overset{\text{sample}}{\mapsto}\end{array} \begin{array}{c}G(\mathbf{x})\\ \boldsymbol{\epsilon}\end{array} \overset{\text{flow}}{\mapsto} \mathbf{z} \overset{\text{generate}}{\mapsto} \hat{\mathbf{x}} \mapsto \ell(\mathbf{x},\hat{\mathbf{x}}) \tag{5.24}$$

As it is simple to draw iid samples and as one typically performs SGD, one often only samples a single instance per pattern in a minibatch.

→ VAE learning of the generative model can be done via backpropagation using samples from the inference model.

Optimizing the ELBO over the variational model and its parameters involves gradients of expectations. However the flow-based representation parameterizes a transformation, which is driven by a fixed noise source such as an iid Gaussian. This is also known as the *re-parameterization* trick. In the case of affine flows, we compute $G(\mathbf{x})$ on the forward pass and then perform Monte Carlo sampling of instances $\boldsymbol{\epsilon}$. For each sample, we can backpropagate through the flow and further through G. This is also known as the *re-parameterization trick* [26].

→ VAE learning of the inference model can be carried out with backpropagation using the reparameterization trick and sampling.

It has been experimentally found that in practice the KL-term effectively performs regularization and that it is advantageous to control its influence with a hyper-parameter. This is also called β-VAE [20]. Finally. we show a diagrammtic view of an VAE in Figure 5.6.

5.4 Generative Adversarial Networks

5.4.1 Generation as Classification

We have seen that maximum likelihood estimation can be used as an inference principle in generative models, but that it comes with drawbacks: It may severely limit the structure and complexity of the admissible transformations as in auto-regressive models and normalizing flows. Or, as seen in the ELBO-based VAE, one may have to resort to approximate inference. On top of this, one may even question not just the *practicability*, but also the *suitability* of likelihood-based inference, if we take indistinguishability of distributions as the goal.

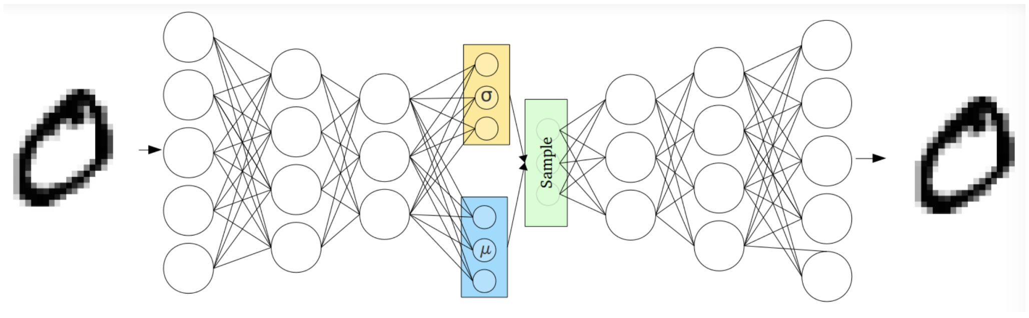


Figure 5.6: Diagrammtic view of a VAE as an autoencoder.

Is the likelihood function (really) the canonical criterion to use in generative modeling?

It has been a seminal idea, first published in [17], to think of generative models in terms of a classification problem. Clearly, if indistinguishability is the goal, then the most natural criteria seems to be the error of a classifier that aims to distinguish between samples drawn from the real distribution $p(\mathbf{x})$ vs. the generative model $p(\mathbf{x};\boldsymbol{\theta})$. This binary classification problem can be formally characterized by the joint distribution

$$p(\mathbf{x},Y;\boldsymbol{\theta}) = \frac{Y}{2}\,p(\mathbf{x}) + \frac{1-Y}{2}\,p(\mathbf{x};\boldsymbol{\theta}), \quad Y \in \{0,1\} \tag{5.25}$$

Here an outcome $Y=1$ refers to the samples generated from the true distribution whereas $Y=0$ refers to the synthesized samples. We have used a balanced approach with equal class probabilities $\mathbb{P}\{Y=1\} = \mathbb{P}\{Y=0\} = \frac{1}{2}$. Let us now assume a classifier – often referred to as *discriminator* – is given, which is a function $\pi : \mathbb{R}^n \to [0;1]$, which we interpret as a prediction for the probability of $\mathbf{x}$ coming from the true distribution, i.e. $\pi(\mathbf{x}) \approx \mathbb{P}\{Y=1|\mathbf{x}\}$. Then we can use the logistic loss (or classification log-likelihood – different from the generative log-likelihood!) as a criterion

$$\ell(\boldsymbol{\theta};\pi) = \mathbf{E}_{\boldsymbol{\theta}}\left[Y \ln \pi(\mathbf{x}) + (1-Y)\ln(1-\pi(\mathbf{x}))\right] \tag{5.26}$$

Note that the dependency on the parameters $\boldsymbol{\theta}$ of the generative model is indirect through the expectation $\mathbf{E}_{\boldsymbol{\theta}}$, which is with regard to the distribution constructed as in Eq. (5.25). For the sake of clarity we can write out explicitly

$$\boldsymbol{\theta} \xrightarrow{\min} \ell(\boldsymbol{\theta};\pi) = \underbrace{\frac{1}{2}\int \ln \pi(\mathbf{x})p(\mathbf{x})d\mathbf{x}}_{\text{constant wrt } \boldsymbol{\theta}} + \underbrace{\frac{1}{2}\int \ln(1-\pi(\mathbf{x}))p(\mathbf{x};\boldsymbol{\theta})d\mathbf{x}}_{\boldsymbol{\theta}\text{-dependent}} \tag{5.27}$$

→ Generative modeling can be directly associated with a (probabilistic) classification problem and its logistic loss.

Example 5.1 Let us try to illustrate this further by assuming that there is a region $U \subseteq \mathbb{R}^n$ such that $\pi(U) > 1-\epsilon$, i.e. where the classifier has a high confidence of samples being 'genuine'. Then the loss in the $\boldsymbol{\theta}$-dependent part is $< \ln(\epsilon) \xrightarrow{\epsilon\to 0} -\infty$. If the generative model succeeds in putting non-vanishing probability mass on U, i.e. generating $\mathbf{x} \in U$, then it can make the loss arbitrarily small in the $\epsilon \to 0$ limit. In the pidgin language of Deep Learning pop culture: the generator can "fool" the discriminator. Thus the discriminator π should be calibrated to not be overconfident about claiming patterns to be non-synthetic. ■

5.4.2 Optimal-Bayes Discriminator

Q What discriminator should be used to judge indistinguishability?

Let us first consider an idealized case, where the classifier is 'perfect', the gold standard for perfection being the Bayes-optimal classifier. We can easily obtain it from Bayes rule as the posterior probability of Y, namely

$$\pi^*(\mathbf{x}) = \mathbb{P}\{Y = 1|\mathbf{x}\} = \frac{p(\mathbf{x})}{p(\mathbf{x}) + p(\mathbf{x};\boldsymbol{\theta})} \tag{5.28}$$

We can insert this formula into the objective in Eq. (5.26) and obtain

$$\begin{aligned}
&\mathbf{E}_{\boldsymbol{\theta}}[Y \ln \pi^*(\mathbf{x}) + (1-Y)\ln(1-\pi^*(\mathbf{x}))] \qquad (5.29)\\
&= \sum_{Y\in\{0,1\}} \int \Big[Y \ln p(\mathbf{x}) + (1-Y)\ln p(\mathbf{x};\boldsymbol{\theta})\Big] \frac{Yp(\mathbf{x}) + (1-Y)p(\mathbf{x};\boldsymbol{\theta})}{2} d\mathbf{x}\\
&\quad - \int \ln(p(\mathbf{x}) + p(\mathbf{x};\boldsymbol{\theta}))\frac{p(\mathbf{x}) + p(\mathbf{x};\boldsymbol{\theta})}{2} d\mathbf{x}\\
&= -\tfrac{1}{2}H(p) - \tfrac{1}{2}H(p(\boldsymbol{\theta})) + H\Big(\tfrac{1}{2}(p + p(\boldsymbol{\theta})\Big) - \ln 2 = \mathrm{JS}(p, p(\boldsymbol{\theta})) - \ln 2
\end{aligned}$$

where JS refers to the Jensen-Shannon divergence between probability distributions, which can also be expressed in terms of Kullback-Leibler divergences between the distributions and their mixture

$$\mathrm{JS}(p,q) = \tfrac{1}{2}\mathrm{KL}(p\,||\,\tfrac{1}{2}p + \tfrac{1}{2}q) + \tfrac{1}{2}\mathrm{KL}(q\,||\,\tfrac{1}{2}p + \tfrac{1}{2}q) \tag{5.30}$$

→ Optimizing the generator so as to maximize the logistic-loss of the Bayes-optimal classifier is equivalent to minimizing the Jensen-Shannon divergence between the true distribution and the model.

This is insightful and has led to various generalizations that use other divergence measures as the criterion. However, note that the analysis and discussion so far relied on the optimal Bayes classifier, which obviously is neither known, nor accessible in many cases of interest.

5.4.3 Learned Discriminators

Q How can we practically define a discriminator to train generative models?

In many cases of practical relevance we would ultimately like humans to be judges of the quality of generated data. But clearly having humans in-the-loop during training is a no-go due to lack of scalability. We thus need to *learn* a classifier and it seems most successful to learn it jointly with the generative model. As we want to harness the power of deep learning, we may decide to also use a DNN for the discriminator, although this is not a necessity. We first note that taking the Bayes-optimal classifier as the reference point, we can $\boldsymbol{\phi}$-parameterize a family of discriminators and get

$$\ell(\boldsymbol{\theta};\pi^*) \geq \sup_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta};\pi(\boldsymbol{\phi})) = \sup_{\boldsymbol{\phi}} \mathbf{E}_{\boldsymbol{\theta}}\left[Y \ln \pi(\mathbf{x};\boldsymbol{\phi}) + (1-Y)\ln(1-\pi(\mathbf{x};\boldsymbol{\phi}))\right] \tag{5.31}$$

This suggests to maximize the criterion with regard to $\boldsymbol{\phi}$, aiming to find the classifier with the lowest logistic loss in expectation. We thus get a saddle point problem

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left\{ \max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta};\boldsymbol{\phi}) \right\}, \quad \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta}^*;\boldsymbol{\phi}) \tag{5.32}$$

This formulation of generative models is also known as Generative Adversarial Networks (GANs). The generator model and discriminator model are thought of as 'adversaries' as they try to optimize a common objective in opposite directions (with opposite sign).

R Under conditions of the Nash's existence theorem for Nash equilibiria (different versions exist) the can also identify

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}.\boldsymbol{\phi}) \tag{5.33}$$

where the order of min and max is no interchanged. In this view, we can also think of GANs as a two-player zero-sum game over continuous action spaces with actions given by the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively.

→ One can define generative modeling as a zero-sum game between two adversaries a generator and a discriminator. This is known as GANs (Generative Adversarial Networks).

One practically relevant issue with the GAN objective arises, when the discriminator – during the process of learning – reliably detects all generated samples as synthetic. Then Eq. (5.27) fails to deliver reliable gradient information (vanishing gradients). This is often circumvented by instead minimizing

$$\boldsymbol{\theta} \overset{\max}{\longrightarrow} \int \ln \pi(\mathbf{x};\boldsymbol{\phi}) p(\mathbf{x};\boldsymbol{\theta}) d\mathbf{x} \tag{5.34}$$

which greatly amplifies the gradient in regions where $\pi(\mathbf{x};\boldsymbol{\phi})$ is close to zero.

5.4.4 Extragradient Method

Q How can we train GANs in the spirit of gradient-based optimization methods like SGD?

GANs are one the few examples in machine learning, where learning can not be cast as a simple minimization/maximization problem. The most naïve approach to try to numerically solve the saddle-point problem is to perform alternating SGD, e.g. in terms of the original objective

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_t, \boldsymbol{\phi}_t), \quad \boldsymbol{\phi}_{t+1} = \begin{cases} \boldsymbol{\phi}_t + \eta \frac{\partial \ell}{\partial \boldsymbol{\phi}}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\phi}_t) & \text{asynchronous} \\ \boldsymbol{\phi}_t + \eta \frac{\partial \ell}{\partial \boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\phi}_t) & \text{synchronous} \end{cases} \tag{5.35}$$

These iterations may converge to a saddle, but this is not guaranteed and it can be shown that one of the failure modes of the asynchronous variant are limit cycles, whereas the synchronous version may in fact diverge (cf. [38]). There is also a wealth of other problems that need to be overcome, often with heuristics or simply with 'tricks of the trade' and tuning.

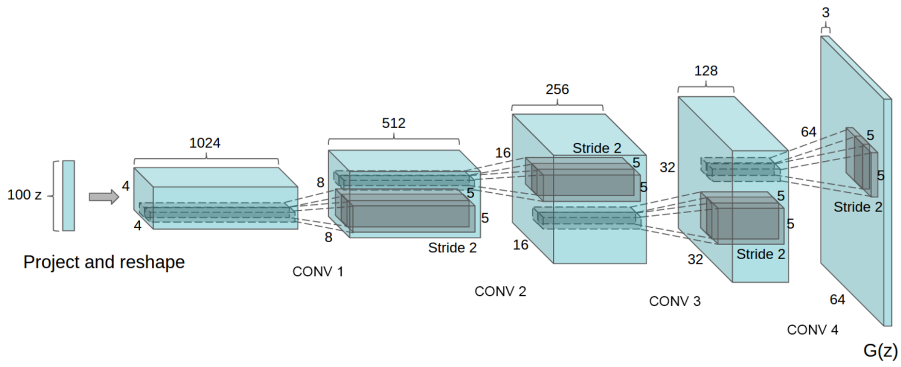


Figure 5.7: DCGAN Architecture

→ GAN training may require a lot of tuning of hyperparameters for models and algorithms. Convergence of learning is often not guaranteed.

There has been much work on how to improve on alternating SGD in a way that convergence guarantees can be made. We here only sketch the idea of extra-gradients invented by [27]and first suggested for GANs in [14]. The idea is compute the gradient at an extrapolated point (like in Nesterov methods) using the synchronous method in Eq. (5.35) and then to update

$$\boldsymbol{\theta}_{t+2} = \boldsymbol{\theta}_t - \eta \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\phi}_{t+1}), \quad \boldsymbol{\phi}_{t+2} = \boldsymbol{\phi}_t + \eta \frac{\partial \ell}{\partial \boldsymbol{\phi}}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\phi}_{t+1}) \tag{5.36}$$

So the main difference is that an intermediate step is made just to get a better estimate of the update direction. The committed iterates are then given by the even iterates $(\boldsymbol{\theta}_{2t}, \boldsymbol{\phi}_{2t})$.

→ There are several modifications of alternating or synchronous gradient descent/ascent that stabilize GAN training, including extragradient methods.

5.4.5 Evaluation

Evaluating (trained) GANs is a non-trivial problem of its own. Note that the value of the objective cannot be used to judge/rank found solutions. Computation of a likelihood score on hold-out data can often only be done in a practically feasible manner using crude approximations. An alternative is the inception score, which can be used for images [38]. One can also compare pairs of generator-discriminator pairs and cross-evaluate the objective [37].

→ Automatic evaluation of GANs remains a challenge and a largely unresolved problem.

5.4.6 Architectures

An early success has been the architecture suggested in [35] ans is known as DCGAN (deconvolutional GANs). A diagrammatic view is shown in Figure 5.7. To get a sense of the quality of generated images, some examples for bedroom images can be seen in Figure 5.8. Recent progress has led to generative models like BigGAN [4] or StyleGAN [23] that synthesize images that are remarkably photorealistic as can be seen from Figure 5.9.

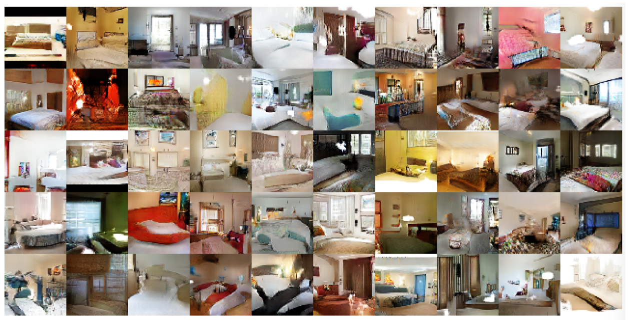


Figure 5.8: Bedroom images generated by DCGAN.

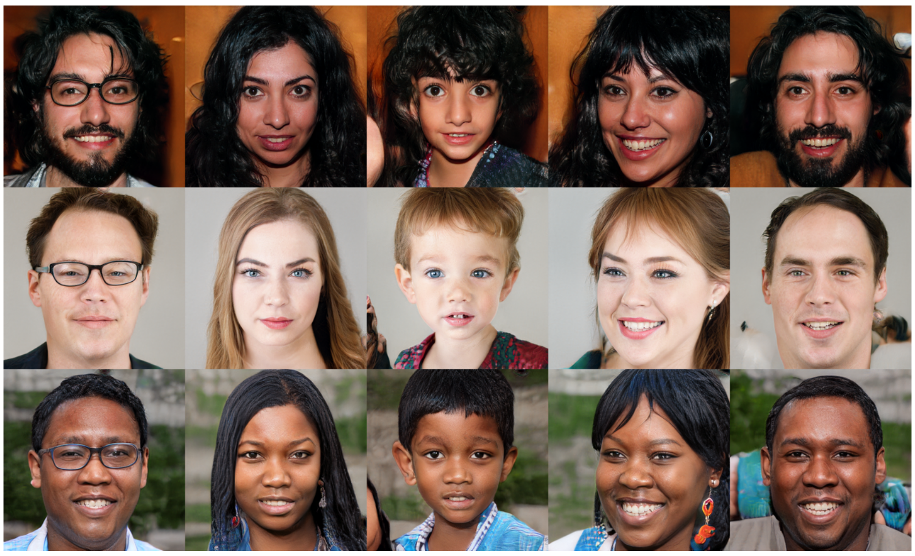


Figure 5.9: Photorealistic face images generated with a StyleGAN model.

Bibliography

[1] Michael W Berry et al. "Algorithms and applications for approximate nonnegative matrix factorization". In: *Computational statistics & data analysis* 52.1 (2007), pages 155–173 (cited on page 62).

[2] David M Blei. "Probabilistic topic models". In: *Communications of the ACM* 55.4 (2012), pages 77–84 (cited on page 59).

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *the Journal of machine Learning research* 3 (2003), pages 993–1022 (cited on page 58).

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018) (cited on page 106).

[5] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. "A singular value thresholding algorithm for matrix completion". In: *SIAM Journal on optimization* 20.4 (2010), pages 1956–1982 (cited on page 40).

[6] Emmanuel J Candès and Terence Tao. "The power of convex relaxation: Near-optimal matrix completion". In: *IEEE Transactions on Information Theory* 56.5 (2010), pages 2053–2080 (cited on page 43).

[7] Hadi Daneshmand et al. "Escaping saddles with stochastic gradients". In: *International Conference on Machine Learning*. PMLR. 2018, pages 1155–1164 (cited on page 83).

[8] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020 (cited on page 8).

[9] Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516* (2014) (cited on page 100).

[10] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803* (2016) (cited on page 100).

[11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011) (cited on page 85).

[12] N Benjamin Erichson et al. "Randomized matrix decompositions using R". In: *arXiv preprint arXiv:1608.02148* (2016) (cited on page 41).

[13] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. "A rank minimization heuristic with application to minimum order system approximation". In: *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148).* Volume 6. IEEE. 2001, pages 4734–4739 (cited on page 40).

[14] Gauthier Gidel et al. "A variational inequality perspective on generative adversarial networks". In: *arXiv preprint arXiv:1802.10551* (2018) (cited on page 106).

[15] Nicolas Gillis and François Glineur. "Low-rank matrix approximation with weights or missing data is NP-hard". In: *SIAM Journal on Matrix Analysis and Applications* 32.4 (2011), pages 1149–1165 (cited on page 36).

[16] Gene H Golub and Charles F Van Loan. "Matrix Computations Johns Hopkins University Press". In: *Baltimore and London* (1996) (cited on page 20).

[17] Ian J Goodfellow et al. "Generative adversarial networks". In: *arXiv preprint arXiv:1406.2661* (2014) (cited on page 103).

[18] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM review* 53.2 (2011), pages 217–288 (cited on page 41).

[19] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pages 770–778 (cited on page 91).

[20] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: (2016) (cited on page 102).

[21] P. Jain, Raghu Meka, and I. Dhillon. "Guaranteed Rank Minimization via Singular Value Projection". In: *NIPS.* 2010 (cited on page 39).

[22] Chi Jin et al. "How to escape saddle points efficiently". In: *International Conference on Machine Learning.* PMLR. 2017, pages 1724–1732 (cited on page 83).

[23] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2019, pages 4401–4410 (cited on page 106).

[24] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (cited on page 86).

[25] Diederik P Kingma and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *arXiv preprint arXiv:1807.03039* (2018) (cited on pages 99, 100).

[26] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013) (cited on page 102).

[27] Galina M Korpelevich. "The extragradient method for finding saddle points and other problems". In: *Matecon* 12 (1976), pages 747–756 (cited on page 106).

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pages 1097–1105 (cited on pages 90, 91).

[29] Jacek Kuczyński and Henryk Woźniakowski. "Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start". In: *SIAM journal on matrix analysis and applications* 13.4 (1992), pages 1094–1122 (cited on page 21).

[30] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pages 788–791 (cited on page 61).

[31] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *arXiv preprint arXiv:1310.4546* (2013) (cited on page 64).

[32] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016) (cited on page 95).

[33] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pages 559–572 (cited on page 9).

[34] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pages 1–17 (cited on page 84).

[35] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015) (cited on page 106).

[36] Danilo Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *International Conference on Machine Learning*. PMLR. 2015, pages 1530–1538 (cited on pages 98, 99).

[37] Kevin Roth et al. "Stabilizing training of generative adversarial networks through regularization". In: *arXiv preprint arXiv:1705.09367* (2017) (cited on page 106).

[38] Tim Salimans et al. "Improved techniques for training gans". In: *arXiv preprint arXiv:1606.03498* (2016) (cited on pages 105, 106).

[39] Badrul Sarwar et al. "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pages 285–295 (cited on page 28).

[40] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *International Conference on Machine Learning*. PMLR. 2016, pages 1747–1756 (cited on page 95).