Learning Linear Assignment Flows for Image Labeling via Exponential Integration

Alexander Zeilmann[1], Stefania Petra[2] and Christoph Schnörr[1]

[1] Image and Pattern Analysis Group, Heidelberg University, Germany
[2] Mathematical Imaging Group, Heidelberg University, Germany

Abstract. We introduce a novel algorithm for estimating optimal parameters of linear assignment flows for image labeling. This flow is determined by the solution of a linear ODE in terms of a high-dimensional integral. A formula of the gradient of the solution with respect to the flow parameters is derived and approximated using Krylov subspace techniques. Riemannian descent in the parameter space enables to determine optimal parameters for a 512×512 image in less than 10 seconds, without the need to backpropagate errors or to solve an adjoint equation. Numerical experiments demonstrate a high generative model expressivity despite the linearity of the assignment flow parametrization.

Keywords: image labeling · assignment manifold · linear assignment flows · parameter learning · exponential integration · low-rank approximation.

1 Introduction

Learning the parameters of large networks from training data constitutes a basic problem in imaging science, machine learning and other fields. The prevailing approach utilizes gradient descent or approximations thereof based on automatic differentiation software tools [4].

In this paper, we focus on a class of networks for image labeling, *linear assignment flows* introduced by [17], where parameter estimation can be based on an *exact* formula of the loss function gradient and its approximation, using established methods of large-scale numerical linear algebra. This enables easy implementations and better control of the approximation error. In particular, neither backpropagation, nor automatic differentiation or solving adjoint equations are required. Our parameter estimation is also time-efficient: for a 512×512 image, the runtime for estimating optimal parameters takes less than 10 seconds. Besides demonstrating these properties, numerical experiments also reveal a high *expressivity* of linear assignment flows: multiscale image structure can be generated from *pure* noise by estimating corresponding parameters.

Related work. Linear assignment flows result from a *linear* tangent space parametrization of the nonlinear *assignment flows* introduced by [3]. While the linear assignment flow is still nonlinear, both the linearity of the corresponding

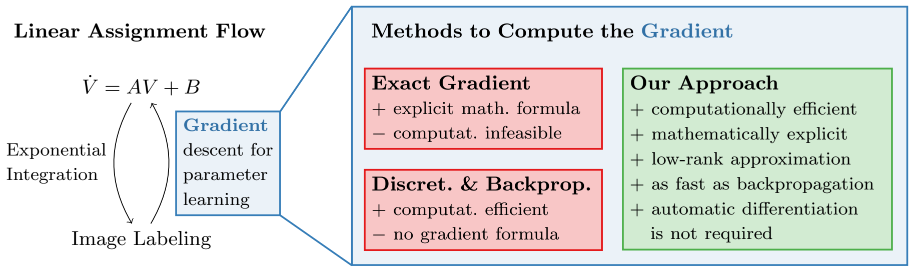

Fig. 1.1. Our approach & related work: pros and cons. Based on the linear assignment flow, our approach performs optimal parameter estimation using a low-rank approximation of the exact gradient. It is as efficient as applying automatic differentiation in frameworks like PyTorch. Unlike this latter methodology, however, our approach is mathematically explicit and hence supports further tasks like flow control.

tangent space ODE and of its parameter dependency facilitate optimal parameter estimation. This has been exploited in [8] by applying symplectic numerical integration for solving both the linear assignment flow and an adjoint ODE for determining the parameter sensitivities.

In this paper, we exploit the fact that the linear tangent space ODE admits a closed-form solution in terms of the high-dimensional integral (Duhamel's formula). By directly approximating an exact gradient formula using established numerical techniques [7,2], error backpropagation can be avoided altogether, and parameter estimation can be done by computing a Riemannian descent flow in the parameter space.

The standard approach to network parameter learning is using automatic differentiation [4]. Automatic differentiation is a general concept applicable to a variety of scenarios. However, the efficient handling of many scenarios requires a more targeted approach. In the current paper, we introduce such a specific algorithm.

The exact integration of the linear assignment flow requires the evaluation of the matrix exponential. The only machine learning frameworks which implement this function are PyTorch [13] and TensorFlow [1]. But even their implementations can not be applied directly to the large matrices required by imaging tasks. In contrast, our approach is able to learn the parameters for very large images or even 3D volumes as commonly found in medical image analysis.

Regarding the numerical techniques employed in our work, related work in scientific computing includes, e.g., [11,9] for computing the Fréchet derivative of the matrix exponential. However, these algorithms are only applicable if the gradient is computed using the *forward* mode of automatic differentiation (i.e., multiplying Jacobians from the right). For functions of the form $\mathbb{R}^n \to \mathbb{R}$ (with large n), the *reverse* mode of automatic differentiation (backpropagation, i.e., multiplying the Jacobians from the left) is significantly more efficient for computing gradients. This is in line with our approach, in which the Jacobians of the loss function and of the additive term of the linear assignment flow are multiplied from the left to the Jacobian of the matrix exponential. This is an essential component for reducing the dimensionality of the problem in our approach.

Contribution. We introduce a novel approach for *learning the model parameters* of the linear assignment flow [17]. These parameters control the regularization properties of image labelings determined by the flow. The problem of *learning* these parameters was raised in [3, Section 5 and Fig. 14]. Our approach has the following properties:

Linear complexity both in storage space and evaluation time with respect to image size and number of labels, which enables to handle large problem sizes.

No automatic differentiation is necessary. While automatic differentiation (and especially backpropagation) has been advancing significantly in recent years, there are not many comprehensive frameworks for it, and its implementation is nontrivial. Our algorithm does not require automatic differentiation and can be conveniently implemented in any numerical software framework.

Explicit parameter gradient formula. While automatic differentiation is a *procedure* computing the gradient, it does generally not lead to a succinct formula. By contrast, our approach is based on an explicit mathematical formula that can be analyzed theoretically and used in future tasks, e.g., for controlling the assignment flow.

Approximating the exact gradient instead of the exact gradient of an approximation. Approaches like exponential integration (see Section 2.3) or explicit Euler integration *approximate* the ordinary differential equation. Automatic differentiation, therefore, produces the exact gradient of this *approximation.* As a consequence, not the parameters of the linear assignment flow are learned but of its approximation. Our approach, on the other hand, directly approximates the exact gradient of the linear assignment flow for parameter estimation and hence enables a better control of the approximation error.

As efficient as backpropagation. In our experiments, our approach was as efficient as backpropagation of exponential integration and backpropagation of explicit Euler. In addition, it requires less storage space.

Low-rank approximation. An approximation is worked out in factorized form. This not only saves storage space but also enables to carry out subsequent computations more efficiently.

Applicability to any loss function and data. Besides being C^1, no further assumptions are made with respect to the loss function used for parameter estimation. Our approach shares with the general assignment flow the property that data in any metric space can be processed.

Organization. Section 2 summarizes the assignment flow, the linear assignment flow and exponential integration for computing it. Section 3 details the exact gradient of any loss function of the flow, with respect to the flow parameters. The application to the linear assignment flow is explained in Section 4 and experimental results are reported in Section 5. We conclude in Section 6.

2 Assignment Flow, Linear Assignment Flow

This section summarizes the *assignment flow* and its approximation, the *linear assignment flow*, introduced in [3] and [17], respectively. The linearized assignment flow provides the basis for our approach to parameter estimation developed in Section 3.

2.1 Assignment Flow

Let $G = (I, E)$ be a given undirected graph with vertices $i \in I$ indexing data $\mathcal{F}_I = \{f_i \colon i \in I\} \subset \mathcal{F}$ given in a metric space $(\mathcal{F}, d)$. The edge set E specifies neighborhoods $\mathcal{N}_i = \{k \in I \colon ik = ki \in E\} \cup \{i\}$ for every vertex $i \in I$ along with positive weight vectors $w_i \in \mathring{\Delta}_{|\mathcal{N}_i|}$, where $\mathring{\Delta}_n = \Delta_n \cap \mathbb{R}^n_{>0}$ denotes the relative interior of the probability simplex Δ_n.

Along with $\mathcal{F}_I$, *prototypical data (labels)* $\mathcal{L}_J = \{l_j \in \mathcal{F} \colon j \in J\}$ are given that represent classes $j = 1, \dots, |J|$. *Supervised image labeling* denotes the task to assign precisely one prototype l_j to each datum f_i at every vertex i in a coherent way, depending on the label assignments in the neighborhoods $\mathcal{N}_i$. These assignments at i are represented by probability vectors

$$W_i \in \mathcal{S} := (\mathring{\Delta}_{|J|}, g_{FR}), \quad i \in I \tag{2.1}$$

on $\mathring{\Delta}_{|J|}$ that endowed with the Fisher-Rao metric g_{FR} becomes a Riemannian manifold denoted by $\mathcal{S}$. Collecting all assignment vectors as rows defines the strictly positive row-stochastic *assignment matrix*

$$W = (W_1, \dots, W_{|I|})^\top \in \mathcal{W} = \mathcal{S} \times \dots \times \mathcal{S} \subset \mathbb{R}^{|I| \times |J|}, \tag{2.2}$$

that we regard as point on the product *assignment manifold* $\mathcal{W}$. Image labeling is accomplished by geometrically integrating the *assignment flow* $W(t)$ solving

$$\dot{W} = R_W\big(S(W)\big), \qquad W(0) = \mathbb{1}_{\mathcal{W}} := \frac{1}{|J|} \mathbb{1}_{|I|} \mathbb{1}_{|J|}^\top \qquad \text{(barycenter)}, \tag{2.3}$$

that provably converges towards a binary matrix [18], i.e., $\lim_{t\to\infty} W_i(t) = e_{j(i)}$, for every $i \in I$ and some $j(i) \in J$, which yields the label assignment $l_{j(i)} \to f_i$. In practice, geometric integration is terminated when $W(t)$ is ε-close to an integral point using the entropy criterion from [3], followed by trival safe rounding [18].

We specify the right-hand side of (2.3) — see (2.5) below — and refer to [3,15] for more details and the background. With tangent space $T_0 = T_p\mathcal{S}$ independent of the base point $p \in \mathcal{S}$, we define

$$R_p \colon \mathbb{R}^{|J|} \to T_0, \qquad z \mapsto R_p(z) = \big(\operatorname{Diag}(p) - pp^\top\big) z, \tag{2.4a}$$

$$\operatorname{Exp} \colon \mathcal{S} \times T_0 \to \mathcal{S}, \quad (p, v) \mapsto \operatorname{Exp}_p(v) = \frac{e^{\frac{v}{p}}}{\langle p, e^{\frac{v}{p}} \rangle} p, \tag{2.4b}$$

$$\operatorname{Exp}^{-1} \colon \mathcal{S} \times \mathcal{S} \to T_0, \quad (p, q) \mapsto \operatorname{Exp}_p^{-1}(q) = R_p \log \frac{q}{p}, \tag{2.4c}$$

where multiplication, division, exponentiation $e^{(\cdot)}$ and $\log(\cdot)$ apply *component-wise* to the vectors. Corresponding maps R_W and Exp_W in connection with the product manifold (2.2) are defined analogously, and likewise the tangent space $\mathcal{T}_0 = T_0 \times \cdots \times T_0$ to $\mathcal{W}$.

The vector field defining the assignment flow (2.3) arises through *coupling* flows for individual pixels through *geometric averaging* within the neighborhoods $\mathcal{N}_i$, $i \in I$, conforming to the underlying Fisher-Rao geometry

$$S(W) = \begin{pmatrix} \cdots \\ S_i(W)^\top \\ \cdots \end{pmatrix} = \mathcal{G}^{\Omega}\big(L(W)\big) \in \mathcal{W}, \qquad \begin{aligned} \Omega &= (\Omega_i)_{i\in I}, \\ \Omega_i &= (\omega_{ik})_{k\in\mathcal{N}_i}, \end{aligned} \tag{2.5a}$$

$$S_i(W) = \mathcal{G}_i^{\Omega}\big(L(W)\big) = \mathrm{Exp}_{W_i}\Big(\sum_{k\in\mathcal{N}_i} \omega_{ik}\, \mathrm{Exp}_{W_i}^{-1}\big(L_k(W_k)\big)\Big), \quad i \in I. \tag{2.5b}$$

The *similarity vectors* $S_i(W)$ are parametrized by *weight patches* $\Omega_i > 0$ that serve as *regularization parameters* and satisfy the constraints $\sum_{k\in\mathcal{N}_i} \omega_{ik} = 1, \forall i \in I$. Flattening these weight patches and complementing zero entries defines sparse row vectors of the matrix $\Omega \in \mathbb{R}^{|I|\times|I|}_{\geq 0}$. Estimating these parameters using the linear assignment flow is the subject of this paper.

2.2 Linear Assignment Flow

The *linear assignment flow*, introduced by [17], approximates (2.3) by

$$\dot{W} = R_W\Big(S(W_0) + dS_{W_0} R_{W_0} \log\frac{W}{W_0}\Big), \quad W(0) = W_0 \in \mathcal{W} \tag{2.6}$$

around any point W_0. In what follows, we only consider the barycenter $W_0 = \mathbb{1}_{\mathcal{W}}$ which is the initial point of (2.3). The differential equation (2.6) is still *nonlinear* but can be parametrized by a *linear* ODE on the tangent space

$$W(t) = \mathrm{Exp}_{W_0}\big(V(t)\big), \tag{2.7a}$$

$$\dot{V} = R_{W_0}\big(S(W_0) + dS_{W_0} V\big) =: B_{W_0} + A(\Omega)[V], \quad V(0) = 0, \tag{2.7b}$$

where the linear operator $A(\Omega)$ linearly depends on the parameters Ω of (2.5) (see [17] for an explicit expression). The linear ODE (2.7b) admits a closed-form solution which in turn enables a different numerical approach (Section 2.3) and a novel approach to parameter learning (Section 3).

2.3 Exponential Integration

The solution to (2.7b) is given by a high-dimensional integral (Duhamel's formula) whose value in closed form is given by

$$V(t;\Omega) = t\varphi\big(tA(\Omega)\big)B_{W_0}, \qquad \varphi(x) = \frac{e^x - 1}{x}, \tag{2.8}$$

where φ is extended to matrix arguments in the standard way [6]. As the matrix A is already very large even for medium-sized images, however, it is not feasible in practice to compute $\varphi(tA)$. Exponential integration [7,12], therefore, was used in [17] for approximately evaluating (2.8).

Applying the row-stacking operator vec_r, that satisfies the general property $\mathrm{vec}_r(ABC) = (A \otimes C^\top)\,\mathrm{vec}_r(B)$, to both sides of (2.7b) and (2.8), respectively, yields with $v = \mathrm{vec}_r(V)$ and the Kronecker product $\otimes$ (cf. [16])

$$\dot{v} = b + A^J(\Omega)v, \qquad v(0) = 0, \qquad b = \mathrm{vec}_r(B_{W_0}), \tag{2.9a}$$

$$A^J = \big(A^J_{ik}\big)_{i,k\in I}, \qquad A^J_{ik} = \begin{cases} \omega_{ik} R_{S_i(W_0)}, & k \in \mathcal{N}_i, \\ 0, & k \notin \mathcal{N}_i. \end{cases} \tag{2.9b}$$

$$v(t;\Omega) = t\varphi\big(tA^J(\Omega)\big)b, \qquad n := \dim v(t) = |I||J|. \tag{2.9c}$$

Using the Arnoldi iteration [14] with initial vector $q_1 = b/\|b\|$, an orthonormal basis $Q_m = (q_1,\dots,q_m) \in \mathbb{R}^{n\times m}$ of the Krylov space $\mathcal{K}_m(A^J, b)$ of dimension m is determined. As reported by [17], choosing $m \leq 10$ yields sufficiently accurate approximations of the actions of the matrix exponential expm and the φ operator on a vector, respectively, that using $H_m = Q_m^\top A^J(\Omega) Q_m$ are given by

$$\mathrm{expm}\big(tA^J(\Omega)\big)b \approx \|b\| Q_m\, \mathrm{expm}(tH_m)e_1, \tag{2.10a}$$

$$t\varphi\big(tA^J(\Omega)\big)b \approx t\|b\| Q_m \varphi(tH_m)e_1, \tag{2.10b}$$

where e_1 denotes the first unit vector. The expression $\varphi(tH_m)e_1$ results from computing the left-hand side of the relation [6, Section 10.7.4]

$$\mathrm{expm}\begin{pmatrix} tH_m & e_1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \mathrm{expm}(tH_m) & \varphi(tH_m)e_1 \\ 0 & 1 \end{pmatrix} \tag{2.11}$$

and extracting the upper-right vector. Since H_m is a small matrix, any standard method [10] can be used for computing the matrix exponential on the left-hand side.

3 Loss Function Gradient and Approximation

Fixing a point of time $t = T$, we set $v_T(\Omega) := v(T;\Omega)$ as given by (2.9c) and assume to be given a loss function

$$\mathcal{L} \colon \mathcal{W} \longrightarrow \mathbb{R}, \qquad \mathcal{L}(\Omega) = f_{\mathcal{L}}\big(v_T(\Omega)\big) \tag{3.1}$$

that evaluates the solution (2.9c) to (2.7b) and hence also the corresponding labeling (2.7a), as a function of the weight parameters Ω; see Section 4 for a concrete example. In this section, various ways to approximate the gradient $\partial\mathcal{L}(\Omega)$ are discussed.

3.1 Loss Function Gradient

We compute differentials of matrix-valued functions $F(X)$ by using the row-stacking operator vec_r: if $\mathrm{vec}_r(F(X)) = f(\mathrm{vec}_r(X))$ defines the vector function f by F, then $\mathrm{vec}_r(dF(X)Y) = df(\mathrm{vec}_r(X))\,\mathrm{vec}_r(Y)$, and we *define* the Jacobian of F by $dF(X) := df(\mathrm{vec}_r(X))$. Thus, if $X \in \mathbb{R}^{m_1 \times m_2}$ and $F(X) \in \mathbb{R}^{n_1 \times n_2}$, then $dF(X) \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}$ and $dF(X)Y \in \mathbb{R}^{n_1 \times n_2}$.

An example is the formula [6, Thm. 10.13] for the vectorized differential of the matrix exponential

$$\mathrm{vec}_r\big(d\,\mathrm{expm}(C)D\big) = \big(\mathrm{expm}(C) \otimes I_n\big)\varphi(-C \oplus C^\top)\,\mathrm{vec}_r(D), \tag{3.2}$$

that we rearranged so as to conform to the row-stacking operator vec_r, rather than to the column-stacking operator used in [6]. $\oplus$ is the Kronecker sum $A \oplus B = A \otimes I_n + I_m \otimes B$ for $A \in \mathbb{R}^{m \times m}$, $B \in \mathbb{R}^{n \times n}$ with identity matrices $I_n \in \mathbb{R}^{n \times n}$.

Proposition 1. *Let $\mathcal{L}$ be a function of the form* (3.1)*, where $v_T(\Omega) = v(T;\Omega)$ solves* (2.9a) *at $t = T$. Then the gradient of $\mathcal{L}$ is given by*

$$\partial\mathcal{L}(\Omega) = \mathrm{vec}_r^{-1}\big(C(\Omega)^\top \partial f_{\mathcal{L}}(v_T(\Omega))\big), \tag{3.3a}$$

$$C(\Omega) = \big((e^{TA^J(\Omega)}, v_T(\Omega)) \otimes e_{n+1}^\top\big)\varphi(-B(\Omega) \oplus B(\Omega)^\top)\,df_B(\mathrm{vec}_r(\Omega)) \tag{3.3b}$$

where $n = |I||J|$, $e_{n+1}^\top = (0_n^\top, 1)$ and $A^J(\Omega)$ is given by (2.9b)*, and*

$$B(\Omega) = \begin{pmatrix} TA^J(\Omega) & Tb \\ 0_n^\top & 0 \end{pmatrix}, \qquad f_B\big(\mathrm{vec}_r(\Omega)\big) = \mathrm{vec}_r\big(B(\Omega)\big). \tag{3.3c}$$

Proof. By (3.1), we have

$$\langle \partial\mathcal{L}(\Omega), U\rangle = df_{\mathcal{L}}(v_T(\Omega))\,dv_T(\Omega)U. \tag{3.4}$$

In order to determine the vector $dv_T(\Omega)U \in \mathbb{R}^n$, we compute analogous to (2.11),

$$\mathrm{expm}\big(B(\Omega)\big) = \begin{pmatrix} \mathrm{expm}\big(TA^J(\Omega)\big) & T\varphi\big(TA^J(\Omega)\big)b \\ 0_n^\top & 1 \end{pmatrix}, \tag{3.5}$$

and note that $v_T(\Omega) = v(T;\Omega)$ given by (2.9c) appears in the last column on the right-hand side. Thus, using $\mathrm{vec}_r(dB(\Omega)U) = df_B(\mathrm{vec}_r(\Omega))\,\mathrm{vec}_r(U)$,

$$\begin{aligned}
v_T(\Omega) &\overset{(3.5)}{=} (I_n, 0_n)\,\mathrm{expm}\big(B(\Omega)\big)e_{n+1} && (3.6a)\\
dv_T(\Omega)U &= (I_n, 0_n)\,d\,\mathrm{expm}\big(B(\Omega)\big)dB(\Omega)Ue_{n+1} && (3.6b)\\
&\overset{(*)}{=} \big((I_n, 0_n) \otimes e_{n+1}^\top\big)\,\mathrm{vec}_r\big(d\,\mathrm{expm}\big(B(\Omega)\big)dB(\Omega)U\big) && (3.6c)\\
&\overset{(3.2)}{=} \big((I_n, 0_n) \otimes e_{n+1}^\top\big)\big(\mathrm{expm}\big(B(\Omega)\big) \otimes I_{n+1}\big) && (3.6d)\\
&\qquad \varphi(-B(\Omega) \oplus B(\Omega)^\top)\,df_B(\mathrm{vec}_r(\Omega))\,\mathrm{vec}_r(U) && (3.6e)\\
&\overset{(3.5)}{=} \big((\mathrm{expm}(TA^J(\Omega)), v_T(\Omega)) \otimes e_{n+1}^\top\big) && (3.6f)\\
&\qquad \varphi(-B(\Omega) \oplus B(\Omega)^\top)\,df_B(\mathrm{vec}_r(\Omega))\,\mathrm{vec}_r(U) && (3.6g)\\
&=: C(\Omega)\,\mathrm{vec}_r(U) && (3.6h)
\end{aligned}$$

where the right-hand side of equation $(*)$ is the vectorization of the left-hand side, i.e., the terms are rearranged, but the overall expression is unchanged.

Since the left-hand side of (3.4) is a matrix inner product, we substitute the last equation into the right-hand side,

$$\langle \mathcal{L}(\varOmega), U\rangle = df_{\mathcal{L}}\big(v_T(\varOmega)\big)C(\varOmega)\operatorname{vec}_r(U) = \big\langle C(\varOmega)^\top \partial f_{\mathcal{L}}(v_T(\varOmega)), \operatorname{vec}_r(U)\big\rangle \tag{3.7}$$

and convert by $\operatorname{vec}_r^{-1}$ the linear form acting on $\operatorname{vec}_r(U)$. $\square$

Expression (3.3) is exact, but its evaluation is computationally infeasible for typical problem sizes. For example, $\varphi(-B(\varOmega) \oplus B(\varOmega)^\top)$ is a dense $(n+1)^2 \times (n+1)^2$ matrix, where $n = |I||J|$ is (number of pixels) $\times$ (number of labels). Therefore, approximations of the loss function gradient $\partial\mathcal{L}(\varOmega)$ are studied next.

3.2 Approximation

Evaluating the gradient (3.3a) requires to multiply a large matrix by a vector,

$$\begin{aligned} C(\varOmega)^\top \partial f_{\mathcal{L}}(v_T(\varOmega)) &= df_B(\operatorname{vec}_r(\varOmega))^\top \varphi(-B(\varOmega)^\top \oplus B(\varOmega))\operatorname{vec}_r\big(F(\varOmega)\big), &\text{(3.8a)}\\ F(\varOmega) &= \begin{pmatrix} \operatorname{expm}(TA^J(\varOmega)^\top) \\ v_T(\varOmega)^\top \end{pmatrix} \Big(\operatorname{vec}_r^{-1}\big(\partial f_{\mathcal{L}}(v_T(\varOmega))\big)\Big) e_{n+1}^\top &\text{(3.8b)}\\ &=: f_1 e_{n+1}^\top. &\text{(3.8c)} \end{aligned}$$

The structure of this expression has the general form [5]

$$f(\mathcal{A})b = f(M_1 \oplus M_2)\operatorname{vec}(B). \tag{3.9}$$

Approximations for large problem sizes exploit the Kronecker sum structure of the matrix valued function f and the assumption that matrix B which generates the vector b has low rank. In our case (3.8a), function φ is applied to a Kronecker sum and the matrix $F(\varOmega)$ (3.8b) has rank 1. Below, we *apply the approach [5] and refine it in terms of an additional approximation* that takes into account the structure of our problem (3.8).

The approach [5] amounts to determine two Krylov subspaces with orthonormal bases $P_m = (p_1, \dots, p_m)$, $R_m = (r_1, \dots, r_m)$ and the standard approximations [6, Section 13.2.1]

$$\begin{aligned} &\mathcal{K}_m(-B(\varOmega)^\top, f_1), & -B(\varOmega)^\top P_m &= P_m T_1 + T_{1;m+1,m} p_{m+1} e_m^\top, &\text{(3.10a)}\\ &\mathcal{K}_m(B(\varOmega), e_{n+1}), & B(\varOmega) R_m &= R_m T_2 + T_{2;m+1,m} r_{m+1} e_m^\top. &\text{(3.10b)} \end{aligned}$$

Setting $U_m = P_m \otimes R_m$ yields

$$\begin{aligned} (-B(\varOmega)^\top \oplus B(\varOmega))U_m &= -B(\varOmega)^\top P_m \otimes R_m + P_m \otimes B(\varOmega)R_m &\text{(3.11a)}\\ &\overset{(3.10)}{=} (P_m \otimes R_m)(T_1 \oplus T_2) + \text{low rank terms} &\text{(3.11b)} \end{aligned}$$

and with $T_m = T_1 \oplus T_2$ and $\mathcal{B}(\varOmega) = -B(\varOmega)^\top \oplus B(\varOmega)$ the approximation of the matrix-vector product on the right-hand side of (3.8a)

$$\begin{aligned} \varphi(\mathcal{B}(\varOmega))\operatorname{vec}_r(F(\varOmega)) &\approx U_m \varphi(T_m) U_m^\top \operatorname{vec}_r(F(\varOmega)) &\text{(3.12a)}\\ &= U_m \varphi(T_m)\operatorname{vec}_r(P_m^\top F(\varOmega) R_m) =: \operatorname{vec}_r(P_m Z R_m^\top), &\text{(3.12b)} \end{aligned}$$

$$Z = \operatorname{vec}_r^{-1}\big(\varphi(T_m)\operatorname{vec}_r(P_m^\top F(\Omega)R_m)\big) \tag{3.13a}$$

$$\overset{(3.8c)}{=} \operatorname{vec}_r^{-1}\Big(\varphi(T_m)\operatorname{vec}_r\big((P_m^\top f_1)(R_m^\top e_{n+1})^\top\big)\Big). \tag{3.13b}$$

The need to store $P_m Z R_m^\top \in \mathbb{R}^{n\times n}$ in the right-hand side of (3.12b) is problematic. Our **additional approximation**, therefore, exploits that fact that the matrix $F(\Omega)$ of (3.8) has rank 1 and that the singular values of $\varphi(T_m)$ typically decrease quickly. As a consequence, we directly approximate $Z \in \mathbb{R}^{m\times m}$ by a singular value decomposition (SVD),

$$Z \approx \sum_{i\in[r]} \sigma_i y_i \otimes z_i^\top, \qquad r \ll m, \qquad \text{to obtain} \tag{3.14}$$

$$\varphi(\mathcal{B}(\Omega))\operatorname{vec}_r(F(\Omega)) \approx \operatorname{vec}_r\Big(P_m\Big(\sum_{i\in[r]}\sigma_i y_i\otimes z_i^\top\Big)R_m^\top\Big) \tag{3.15a}$$

$$= \operatorname{vec}_r\Big(\sum_{i\in[r]}\sigma_i(P_m y_i)\otimes(R_m z_i)^\top\Big) = \sum_{i\in[r]}\sigma_i(P_m Y_i)\otimes(R_m Z_i). \tag{3.15b}$$

The last expression shows that $U_m = P_m \otimes R_m$ does not have to be explicitly computed and stored. By leaving the Kronecker product unevaluated, the resulting vector (3.15b) can be conveniently multiplied with the Jacobian $df_B(\operatorname{vec}_r(\Omega))^\top$ of (3.8a). In addition, merely $\mathcal{O}(2rn)$ numbers have to be stored which is feasible even for large problem sizes like 3D image labeling problems.

3.3 Computing the Gradient using Automatic Differentiation

An entirely different approach for computing the gradient $\partial\mathcal{L}(\Omega)$ of the function (3.1) is using automatic differentiation. To this end, we implemented the explicit Euler integration of the linear assignment flow (2.7b)

$$V^{(k+1)} = V^{(k+1)} + h\Big(A(\Omega)[V^{(k)}] + B_{W_0}\Big), \qquad V^{(0)} = 0 \tag{3.16}$$

in PyTorch and used PyTorch's automatic differentiation capabilities to compute the gradient. Similarly, we let PyTorch compute the gradient of the exponential integration (2.10b) of the linear assignment flow.

From a numerical point of view, we compare the approaches in Section 5.

4 Application to the Linear Assignment Flow

We provide further details on our implementation.

Loss Function. Because we can use trivial rounding to convert tangent vectors $V \in \mathcal{T}_0$ and in turn assignment matrices $W \in \mathcal{W}$ to a labeling, only the direction of the tangent vectors is important, but not their length. Thus, we consider the angle between the solution of the linear assignment flow V and a ground truth direction $V^* = \operatorname{Exp}_{\mathbb{1}_{\mathcal{W}}}^{-1}(W^*)$, with ground truth labeling W^*, as loss function

$$f_{\mathcal{L}}\colon \mathbb{R}^n \longrightarrow \mathbb{R}, \qquad V \longmapsto 1 - \frac{\langle V^*, V\rangle}{\|V^*\|\|V\|}. \tag{4.1}$$

We point out that our approach works with any C^1 loss function.

Riemannian Gradient Descent. As stated in Section 2.1, the parameters of the weight patches as nonzero entries of the row-stochastic matrix Ω of the linear assignment flow can be represented in the same way as the assignment vectors on the assignment manifold $\mathcal{W}$. Consequently, we convert the Euclidean gradient $\partial\mathcal{L}(\Omega)$ from Section 3 into the Riemannian gradient $\nabla\mathcal{L}(\Omega) = R_{\Omega}\partial\mathcal{L}(\Omega)$ using the mapping (2.4a) and perform Riemannian gradient descent

$$\Omega^{(k+1)} = \exp_{\Omega^{(k)}}(-h\nabla\mathcal{L}(\Omega^{(k)})), \quad k = 0, 1, 2, \dots \tag{4.2}$$

with step size $h > 0$. As initialization, we choose uniform weights $\Omega^{(0)} = \mathbb{1}_{\mathcal{W}}$.

Parameter Influence. Our implementation also takes into account the dependency of the vector b of (2.9a) and the block-matrices $R_{S_i(W_0)}$ of (2.9b) on $\Omega_i = (\omega_{ik})$. Due to lack of space, we did not include formulas of the corresponding gradient components in Section 3.

5 Experiments

Figures 5.1, 5.2 and 5.3 illustrate the applicability and performance of our approach. We refer to the captions for details.

A comparison of our algorithm and the two algorithms using backpropagation is depicted by Figure 5.2. Generally speaking, regarding numerical computations (Fig. 5.2, left), the difference between the three algorithms is negligible compared to the influence of other hyperparameters like step size (learning rate) h for the gradient descent or neighborhood size $|\mathcal{N}_i|$, respectively. Our algorithm, however, has further advantageous properties as listed on page 3. In particular, it explicitly returns a subspace of low dimension m (Fig. 5.2, left) that will be useful for further tasks related to label prediction and flow control.

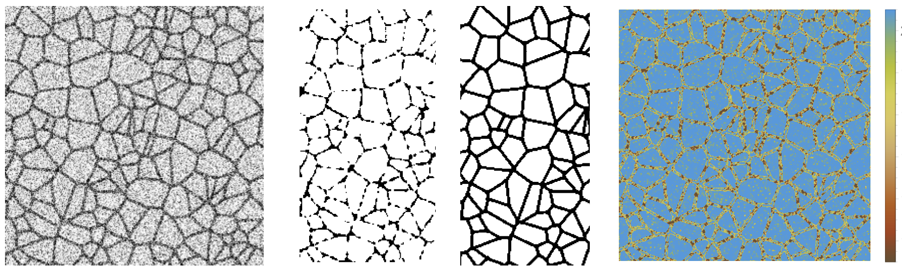

Fig. 5.1. Left: Noisy artificial vessel-like structures used as input data. **Center-left:** Left half of the labeling (labels: $J = \{\square, \blacksquare\}$) result using *uniform* weights. **Center-right:** The labeling result based on *learned* weights is very close to the ground truth ($< 0.1\%$ wrong pixels). **Right:** Pseudo-color plot of entropies of learned weight patches for each center pixel. The algorithm learned where to denoise with uniform weights (high entropy) and where to enforce structure using non-uniform weights (low entropy).

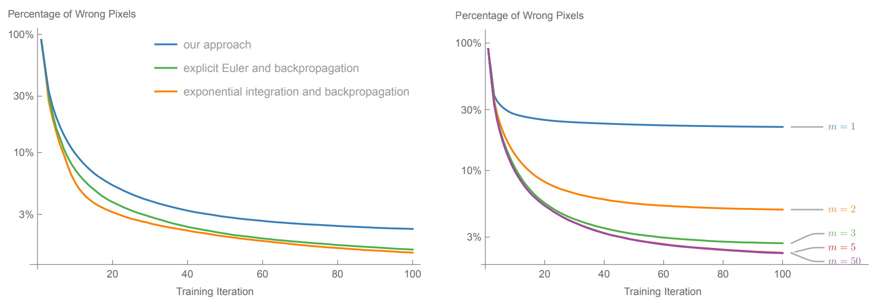

Fig. 5.2. Comparison of the loss for different algorithms and Krylov subspace dimensions. Left: We chose the Krylov subspace dimension resp. the discrete time step size such that all three algorithms required about the same computation time. Closeness of the curves demonstrates that all methods estimate the parameters accurately. The slightly worse accuracy of our algorithm is due to the fact that it uses an approximation of the gradient instead of the exact gradient of an approximated model. Our approach, however, is mathematically explicit and enjoys the pros displayed by Figure 1.1. **Right:** For our approach to parameter learning based on the linear assignment flow, a Krylov subspace dimension of $m = 5$ turned out to sufficient for most experiments, as illustrated by the plot.

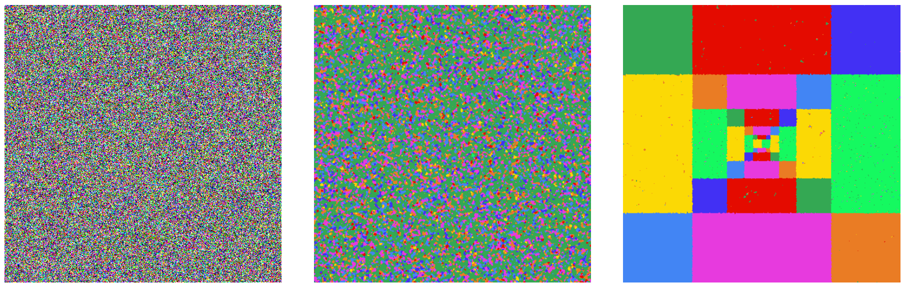

Fig. 5.3. Expressivity of the linear assignment flow. Left: Pure noise used as input data. **Center:** Labeling ($J = \{$🟦, 🟦, 🟩, 🟩, 🟨, 🟧, 🟥, 🟪$\}$) with uniform weights returns more than 90% wrongly labeled pixels. **Right:** Our learning approach takes less than 10 seconds and produces a labeling at multiple spatial scales with well below 1% wrongly labeled pixels. These wrong label assignments can be corrected by choosing a larger neighborhood size or by iteratively applying the assignment flow again.

6 Conclusion

We presented a novel approach for learning the parameters of the linear assignment flow for image labeling. The algorithm approximates the computationally infeasible exact gradient of a linear dynamical system with respect to the regularizing weight parameters, using exponential integration, low-rank approximation and sparse matrix-vector multiplications. Regarding runtime, our research implementation is on par with highly tuned-machine learning toolboxes, and it additionally returns the essential information for image labeling in terms of a low-dimensional parameter subspace. Our future work will study on this basis problems related to label prediction and flow control.

Acknowledgments

This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2181/1 - 390900948 (the Heidelberg STRUCTURES Excellence Cluster).

References

1. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems. In: OSDI (2016)
2. Al-Mohy, A.H., Higham, N.J.: Computing the Fréchet Derivative of the Matrix Exponential, with an Application to Condition Number Estimation. SIAM J. Matrix Anal. Appl. **30**(4), 1639–1657 (2009)
3. Åström, F., Petra, S., Schmitzer, B., Schnörr, C.: Image Labeling by Assignment. Journal of Mathematical Imaging and Vision **58**(2), 211–238 (2017)
4. Baydin, A., Pearlmutter, B., Radul, A., Siskind, J.: Automatic Differentiation in Machine Learning: a Survey. J. Machine Learning Research **18**, 1–43 (2018)
5. Benzi, M., Simoncini, V.: Approximation of Functions of Large Matrices with Kronecker Structure. Numerische Mathematik **135**(1), 1–26 (2017)
6. Higham, N.J.: Functions of Matrices: Theory and Computation. SIAM (2008)
7. Hochbruck, M., Lubich, C.: On Krylov Subspace Approximations to the Matrix Exponential Operator. SIAM J. Numer. Anal. **34**(5), 1911–1925 (1997)
8. Hühnerbein, R., Savarino, F., Petra, S., Schnörr, C.: Learning Adaptive Regularization for Image Labeling Using Geometric Assignment. Journal of Mathematical Imaging and Vision **63**, 186–215 (2021)
9. Kandolf, P., Koskela, A., Relton, S.D., Schweitzer, M.: Computing Low-Rank Approximations of the Fréchet Derivative of a Matrix Function Using Krylov Subspace Methods. arXiv:2008.12926 (2020)
10. Moler, C., Loan, C.V.: Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. SIAM Review **45**(1), 3–49 (2003)
11. Najfeld, I., Havel, T.F.: Derivative of the Matrix Exponential and Their Computation. Adv. Appl. Math. **16**(3), 321–375 (1995)
12. Niesen, J., Wright, W.M.: Algorithm 919: A Krylov Subspace Algorithm for Evaluating the φ-Functions Appearing in Exponential Integrators. ACM Transactions on Mathematical Software **38**(3), 1–19 (2012)
13. Paszke, A., et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: NIPS (2019)
14. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM (2003)
15. Schnörr, C.: Assignment Flows. In: Grohs, P., Holler, M., Weinmann, A. (eds.) Variational Methods for Nonlinear Geometric Data and Applications, pp. 235–260. Springer (2020)
16. Van Loan, C.F.: The Ubiquitous Kronecker Product. J. Comput. Appl. Math. **123**, 85–100 (2000)
17. Zeilmann, A., Savarino, F., Petra, S., Schnörr, C.: Geometric Numerical Integration of the Assignment Flow. Inverse Problems **36**(3) (2020)
18. Zern, A., Zeilmann, A., Schnörr, C.: Assignment Flows for Data Labeling on Graphs: Convergence and Stability. arXiv:2002.11571 (Feb 20, 2020)