Objects Flowing Through Videos

Lukas Bold [* 1]   **Sebastian Damrich** [* 1]

Abstract

In this work, we explore ways of tracking objects using a Minimum-Cost Flow Problem (MCFP) whose parameters are the output of a neural network. We find that differentiating through the linear problem with ideas from the OptNet paper (Amos & Kolter, 2017) is difficult, propose workarounds and implement one using shortest-paths. Moreover, we discuss a differentiable approach based on (Cuturi, 2013)'s entropy regularised optimal transport problem.

1. Introduction

In this report, we discuss strategies for tracking an object in a video given the initial position of the object. At training time the position of the object in more frames is available. The main idea of all our approaches is to treat the movement of the object as the flow in a graph that is built from connecting the pixels of one frame of the video with the pixels of the next frame. The pixels through which there is flow shall indicate the position of the object. The flow is computed by a variant of the MCFP whose parameters are the output of a neural network. We follow two main trains of thought, which were both suggested to us by Prof. Hamprecht. In the first one, we examine ways of differentiating through the optimisation problem of the MCFP, see section 3. We explain difficulties of applying ideas along the lines of (Amos & Kolter, 2017) to our problem in section 3.3 and introduce workarounds for special cases of the MCFP in section 3.4. In section 5 we describe the implementation of one of those workarounds in more details and present the results in section 6.2.

The second train of thought considers a special version of the MCFP, namely a version of optimal transport, which we discuss in section 4. Here, we do not try to differentiate through the optimisation problem itself, but differentiate through the steps of Sinkhorn's algorithm which solves an entropy regularised problem.

---

[*]Equal contribution [1]University of Heidelberg, Heidelberg, Germany. Correspondence to: Lukas Bold <lukasbold92@gmail.com>, Sebastian Damrich <sebastian.damrich@gmx.de>.

2. Related Work

Min-cost flows are popular for multi-target tracking problems (Zhang et al., 2008). Similar to one of our approaches shortest paths algorithms have been used to reduce the computation time of MCFPs ((Berclaz et al., 2011), (Lenz et al., 2015)). (Schulter et al., 2017) smooth out the MCFP in order to differentiate the flow by the costs. So, their work is similar to our objective. The main difference, however, is that all the above tracking approaches consider multiple objects which have been detected in a prior step. The graph in which they compute min-cost flows only links up these detections and is thus much smaller than in our setting, where finding the location of the object in late frames is the main aim.

(Ochs et al., 2015) address the general problem of bi-level optimisation problems with a non-smooth lower layer. They propose to differentiate an iterative solution algorithm of the lower layer problem. This has resemblance with the optimal transport idea, which we explore in section 4.

3. Differentiating through a MCFP

Our first strategy is to model the movement of the object in the video as a flow which is minimal with respect to some costs. Then we try to differentiate through the MCFP to improve the costs.

3.1. Min-Cost-Flow programs

Given a directed graph $G = (V, E)$ with edge weights c, edge capacities κ and vertex supply b, the MCFP is the problem of finding a flow of minimal cost along the edges of the graph, which respects the capacity constraints and manages to match up the vertices with positive and negative flow supply. This can be formulated as a linear program:

$$\min_f \sum_{e \in E} c(e)\, f(e) \text{ s.t.}$$

$$\forall v \in V : \sum_{(v,w) \in E} f(v, w) = \sum_{(u,v) \in E} f(u, v) + b(v) \quad (1)$$

$$\forall e \in E : 0 \leq f(e) \leq \kappa(e).$$

The first condition ensures that flow is conserved and the supply and demand at each vertex are met while the second condition guarantees the capacity constraints. In the

following, we will often write c, b, κ and f as vectors of length $|E|$.

3.2. MCFP for object tracking

We construct the directed graph on which we want to compute a min-cost-flow as follows: The vertices are given by the pixels of the video in addition to an auxiliary node t, which serves as a sink. There are edges from a pixel to a square area of pixels around it in the next frame (pixels close to the borders of the frame have less outgoing edges). We call the length of the sides of this square window size. It should be an odd number. All the pixels in the last frame have an edge to the sink. The flow supply is one for every pixel in the first frame that belongs to the object. This encodes that we know the position of the object in the first frame. All other pixels have a flow supply of zero apart from t which has a flow demand matching the cumulative flow supply.

The costs and capacities of the the graph are the output of a neural network, which takes the video as input.

Given a min-cost flow for the graph described above, we predict the location of the object in every frame by the pixels through which there is a positive flow. We can also associate a loss ℓ to a flow. This is done by subtracting the the amount of flow that goes through the object in the last frame from the amount of flow that misses the object in the last frame. As a result, we could capture the change of the loss by the flow by the gradient:

$$\left(\frac{\partial \ell}{\partial f}\right)_{(v,w)} = \begin{cases} 1 - 2 \cdot \mathbb{1}(v \in \text{object}) & \text{if } w = t \\ 0 & \text{else} \end{cases} \quad (2)$$

In order to train the neural network, we need to know $\frac{\partial \ell}{\partial c}$ and $\frac{\partial \ell}{\partial \kappa}$. Given $\frac{\partial \ell}{\partial f}$ as above and using the chain rule, the key is to find $\frac{\partial f}{\partial c}$ and $\frac{\partial f}{\partial \kappa}$. In other words, we need to find out how the optimal flow, the solution to the linear program (1), changes with the problem parameters c and κ. For this task we explored the ideas of (Amos & Kolter, 2017). If at training time the position of the object is known in more than the first and last frame, one could include them into the loss function. If the position of the object in the last frame is known at test time, one could omit the sink and uniformly distribute the flow demand over the object's position in the last frame and predict the location in the intermediate frames. In this chapter, we focus on the situation where the location of the object in the first and last frame is known at training time but only the initial position is known at test time.

3.3. Differentiating through KKT Conditions

The OptNet Paper (Amos & Kolter, 2017) introduced a method for differentiating through a quadratic problem

given by

$$\min_{f} \frac{1}{2} f^T Q f + c^T f$$

$$\text{subject to } Af - b = 0 \text{ and } Gf \leq h,$$

where $f \in \mathbb{R}^n$ is the optimization variable, $Q \in \mathbb{R}^{n \times n}$ is a positive semidefinite matrix, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{p \times n}$ and $h \in \mathbb{R}^p$ are the data describing the minimization problem.

Since our first aim was the application of this process to the MCFP described above, we could restrict ourselves to a linear optimization problem of the following form:

$$\min_{f} c^T f$$

$$\text{subject to } Af = b, \ f \geq 0 \text{ and } f \leq \kappa,$$

where f is the flow through the graph, c is the cost vector and κ is the capacity vector related to the graph's edges. Together with suitable A, b and the inequalities $f \geq 0$, $f \leq \kappa$, this case encodes the MCFP. Now we apply the ideas of the OptNet Paper to this linear problem and use the definitions $G := \begin{pmatrix} -I \\ I \end{pmatrix}$ and $h := \begin{pmatrix} 0 \\ h \end{pmatrix}$

to get the both restrictions $Af = b$ and $Gf \leq h$. Then we obtain the Lagrangian function

$$L(f, \nu, \lambda) = c^T f + \nu^T(Af - b) + \lambda^T(Gf - h)$$

with $\lambda \geq 0$ which leads us to the KKT conditions

$$\nabla_f L(f^*, \nu^*, \lambda^*) = c + A^T \nu^* + G^T \lambda^* = 0$$
$$\nabla_\nu L(f^*, \nu^*, \lambda^*) = Af^* - b = 0$$
$$D(\lambda^*)(\nabla_\lambda L(f^*, \nu^*, \lambda^*)) = D(\lambda^*)(Gf^* - \kappa) = 0$$

where D(·) creates a diagonal matrix consisting of the entries of a vector and f^*, ν^* and λ^* are the optimal primal and dual variables. After taking the differentials of the KKT conditions we receive the following matrix equation:

$$\begin{pmatrix} 0 & G^T & A^T \\ D(\lambda^*)G & D(Gf^* - h) & 0 \\ A & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathrm{d}f \\ \mathrm{d}\lambda \\ \mathrm{d}\nu \end{pmatrix} = \quad (3)$$
$$\begin{pmatrix} -\mathrm{d}c - \mathrm{d}G^T \lambda^* - \mathrm{d}A^T \nu^* \\ -D(\lambda^*)\mathrm{d}Gf^* + D(\lambda^*)\mathrm{d}h \\ -\mathrm{d}Af^* + \mathrm{d}b \end{pmatrix}$$

where the left hand matrix is from now on called M. In the next step, we are interested in the derivative $\frac{\partial \ell}{\partial c} \in \mathbb{R}^n$, where $\ell : \mathbb{R}^n \to \mathbb{R}$, $f \mapsto \ell(f)$ is our loss function. So we want to know how the loss varies with respect to the costs of the graph. Later, we also discuss the approach of considering $\frac{\partial \ell}{\partial h} \in \mathbb{R}^n$, where h carries the information about

the capacities κ. After replacing d by $\frac{\partial}{\partial c}$ in equation (3) we obtain $M(\frac{\partial f}{\partial c}, \frac{\partial \lambda}{\partial c}, \frac{\partial \nu}{\partial c})^T = (-I, 0, 0)^T$. If the derivation by h is desired, the right hand side has to be replaced by $(0, I, 0)^T$. For the next point, it is necessary to consider the construction of A in a few words. The rows of A are indexed by the number of pixels in the video plus one vertex for the sink and the columns by the edges in the video's graph. Most entries of A are zeros. In each column there are one 1 and one -1 indicating source and target of the edge. So A is the incidence matrix of the graph and its rank is therefore the number of rows minus 1.

In our case the top left entry of M is the zero matrix and A does not have full rank in contrast to the positive definite matrix Q and full rank matrix A in the OptNet paper in Theorem 1. Therefore, it might happen, that the above equation has no solution at all or a non-unique one, because our matrix M is not invertible. However, we can continue and try to compute a solution (if it exists), which is of course not unique. With this solution for $\frac{\partial f}{\partial c}$ and derivative $\frac{\partial \ell}{\partial f}$, computed above, we could compute the desired derivation $\frac{\partial \ell}{\partial c} = \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial c}$ to insert it as the gradient for the neural network.

3.3.1. THEORETICAL ISSUES

The first issue was already mentioned above. We don't know if an analytic solution (x, y, z) for equation (3) exists. If so, there would be infinitely many solutions $(x, y, z) + \ker M$ satisfying this equation. This could become a problem if iterated computations of these solutions with some varied parameters are necessary. The iterated solutions might not go in the same "direction", so that this could lead to unexpected effects.

One possible idea might be to add a "small" quadratic term Q and to change A to a full rank matrix in a way that does not change M in a too dramatically way to obtain a uniquely solvable problem.

If the system has no exact solution, one can use for example the least squares solution to get results which we can continue working with.

The feasable set of a linear program is a polyhedron and (unique) solutions lie on the vertices of that polyhedron. This is why the function $f : \mathbb{R}^n \to \mathbb{R}^n$, which yields the solution for the optimization problem, is hard to differentiate by c as the solution either remains the same or jumps to a different vertex. Maybe it is possible to add a regularisation term to make the objective function strictly convex and thus easier to differentiate. A similar approach using an entropic regularisation term is mentioned in (Cuturi, 2013).

In contrast to the method above, differentiating by capacities has more continuous properties. If the capacities are changed, the edges of the feasible polyhedron are shifted, which is a continuous process.

It should also be referred to the fact that f itself is not a map

in general. Since f is a solution of a linear program, there might be more than one optimal flow. However, in practice f is considered as the flow through a video's graph, so we hope that the different optimal flows don't cause too diverging results.

3.3.2. PRACTICAL ISSUES

The main practical issue is the huge size of the problem. The OptNet paper states that more than 1000 parameters are infeasable. Even when shrinking and cropping the original videos in our problem there is no possibility for an acceptable computation time. To be precise, the matrix A has shape (# pixels in the video + 1, # edges of the graph) where the number of edges in the graph is roughly equal to $[(\text{window size})^2 * (\text{# frames in the video} - 1) + 1] * \text{width} * \text{height}$. This was way too much for our computational capacities, even if we used an extremely shrunk video. Maybe working with sparse matrices could improve the performance, since A is a very sparse matrix. We have not explored it yet, but we expect the problem size still being too big.

Additionally, the QP Solver provided by the OptNet authors did not work, so we were not able to compare their performance and computational results with ours. It also could be possible, that their QP Solver is not even capable of solving our problem without adding a positive definite matrix Q.

3.4. Special cases of the MCFP

We saw in the above section, that the approach of (Amos & Kolter, 2017) seems not well suited for our problem. Therefore, we did not try to differentiate through the optimisation problem to obtain a gradient for training the neural networks. Instead, we used a solver to solve the MCFP problem and then updated the parameters of the problem in a way that reduces the loss at least heuristically at the solution. Those updated parameters then serve as targets for the neural network. In our analysis, we specialised the general MCFP to two corner cases. This reduces the computation time and highlights different features of costs and capacities.

3.4.1. SHORTEST PATHS

We explored the situation without upper capacity constraints in some detail. Without upper capacity constrains, the MCFP becomes the problem of finding the shortest paths from the pixels that constitute the object in the first frame to the sink. One advantage of this approach is that there are fast solvers for Shortest Paths problems, such as the Dijkstra or the Bellman-Ford algorithm.[1] Those paths

---

[1] Due to the special directed structure of our graph, finding shortest paths can be computed by the Dijkstra algorithm (and

that do not go through object in the last frame, have lost the object at some point. Therefore, we penalise them by increasing the cost of the edges along such a path by a penalty parameter. One the other hand, paths that go through the object in the last frame are rewarded. The costs along their edges is reduced by the penalty parameter.

The shortest paths algorithm gives the shortest paths from the background of the first picture to the sink for free. In order to increase the amount of information passed to the neural network, we also update the costs along the paths from the background to the sink in the same fashion as before, but in opposite direction: Paths from the background that pass through the background of the last frame are rewarded and those that pass through the object in the last frame are penalised.

One issue with this way of updating costs is that if a path that is penalised and one that is rewarded have a common end (they start in the object and the background respectively, but merge at some point), the rewards and penalties on the common end cancel. The possibility of the merging of paths itself could also be an issue. If an edge has very small cost, it might induce several paths to use this edge and hence to merge. Moreover, without upper bounds on the capacities there is no possibility for two paths to separate once they have merged. This would be particularly problematic when the object magnifies during the video. In general, it might happen that many paths merge at some point and that as a result the last frame is only passed at very few pixels. This would yield degenerated predictions of the objects location. We implemented this approach (section 5). When testing it on a real video (section 6.2) we fortunately observed only little over-merging.

Dijkstra's algorithm can be seen as a version of message passing and our graph is cycle-free. Thus, one could also try to differentiate the flow with respect to the costs by unrolling the message passing iterations over time and differentiating them.

3.4.2. MAXIMUM FLOW

The other extreme case of a MCFP would be to focus entirely on the capacities rather than the costs. By setting the costs to zero for all edges, the cost-minimisation problem becomes ill-defined. In order to dodge this problem, one could replace the cost-minimisation problem with a flow maximisation problem. Instead of defining a fixed flow supply for the nodes in the first frame and the sink, one could allow the pixels that constitute the object in the first frame to send as much flow as possible and the sink to take in as much flow as possible. For all other pixels flow-in equals flow-out still applies. The resulting linear program

_____

thus a version of message passing) even when costs are negative.

would look like this

$$\max_f \sum_{(v,t)\in E} f(v,t) \text{ s.t.}$$

$$\forall v \in V \setminus (V' \cup \{t\}) \sum_{(v,w)\in E} f(v,w) = \sum_{(u,v)\in E} f(u,v)$$

$$(4)$$

$$\forall e \in E : 0 \leq f(e) \leq \kappa(e)$$

where V' are the pixels in the first frame that constitute the object. One advantage of such a simplification of the MCFP is that there are several fast algorithms for solving a maximum flow problem, for instance the Ford-Fulkerson (or Edmund-Karp) algorithm or the Pre-Flow-Push algorithm. So as above, one could let a neural network predict capacities given the video, then find a maximum flow through the resulting network and finally update the capacities in a meaningful way to provide an update direction for the neural network. Similar to the case of shortest paths, one way to update the capacities is to increase the capacities along a path of positive flow to the object in the last frame and to decrease the capacities (bearing in mind the lower bound of zero capacity) along a path of positive flow to a background pixel in the last frame. A theoretical appeal of this strategy is that it nicely matches up with the notion of augmenting paths that are used to determine a maximum flow with the Ford-Fulkerson algorithm.

Another advantage of using capacities rather than costs is that maximum flow behaves more continuously and piecewise even differentiably with regard to the capacities in contrast to the case of a change in the costs. Changing the capacity of an edge on which the flow is not maximal infinitesimally, does not change the flow at all. Decreasing the capacity of an edge on which the flow is maximal, reduces the flow on that edge with a derivative of one. Increasing the capacity on such an edge either increases the flow on that edge with derivative one (if the network allows more flow to enter and leave the edge) or it does not change the flow on that edge at all. The effect on other edges can be manifold, due to the non uniqueness of a maximal flow on some networks, however there always are maximal flows that change piecewise differentially on all edges with respect to the capacities.

Normalising the loss from section 3.2 by the size of the flow and pretending that changing the capacity of an edge with maximal flow always results in a derivative of one yields a crude approximation to a derivative $\frac{\partial \ell}{\partial \kappa}$:

$$\left(\frac{\partial \ell}{\partial \kappa}\right)_{(v,w)} = \begin{cases} \frac{1-2\cdot \mathbb{1}(w\in \text{object})}{\sum_{(u,t)} f(u,t)} & \text{if } w \text{ is in last frame and} \\ & f(v,w) = \kappa(v,w) \\ 0 & \text{else} \end{cases}$$

However, this is more or less just updating the capacities of edges leading towards the last frame depending on whether

they lead to the object or not. This latter approach however seems to resemble more an object detection scheme than an object tracking one.

An upside of using capacities is that in contrast to using shortest paths capacities can handle an object which increases in size well.

4. Optimal transport via the Sinkhorn algorithm

In this section we explore another variant of the MCFP for object tracking. We model the object's location as a probability distribution over the set of pixels of a frame. Given the first and last position of the object we want to predict its position in the intermediate frames. With distributions d^t and d^{t+1} for the position of the object in two consecutive frames and a cost matrix C for transporting flow between any pixel in one frame to any pixel in the other frame, one can compute the Earth Mover's distance between them, which is the following linear problem

$$\min_P \langle P, C \rangle \text{ s.t.} \qquad (5)$$
$$P^T \cdot \mathbb{1} = d^t, P \cdot \mathbb{1} = d^{t+1}$$
$$\forall i, j : P_{ij} \geq 0$$

where $\langle P, C \rangle = \sum_{i,j} P_{ij} C_{ij}$. This can be seen as a transport problem and as such as a MCFP without upper capacity constraints. The linear problem is infeasable to solve for large dimensional distributions as in our case. In order to speed up computation, (Cuturi, 2013) adds an entropy regularisation term of the form $-\frac{1}{\lambda} h(P) := \frac{1}{\lambda} \sum_{i,j} P_{ij} \log(P_{ij})$ to the objective function. The solution of this entropy regularised version of the problem has a special form, due to which it can quickly be computed using Sinkhorn's algorithm ((Sinkhorn, 1967) and (Cuturi, 2013) Lemma 2).

In our situation, we consider one distribution for each frame of the video, $d^1, \ldots, d^N$, of which d^1 and d^N are given as uniform distributions over the object's location in the first and last frame. The others are unknown. As before, we let a neural network compute the costs from the video. Let C^t denote the cost for transporting flow between the pixels of frame t and $t+1$. Candidates for the object's position in all frames, could be those distributions $d^2, \ldots, d^{N-1}$ for which the optimal transport is minimal:

$$\min_{\{P^t\}_{t=1}^{N-1}, \{d^t\}_{t=2}^{N-1}} \sum_{t=1}^{N-1} \langle P^t, C^t \rangle \text{ s.t.} \qquad (6)$$
$$\forall t : P^{t,T} \cdot \mathbb{1} = d^t, P^t \cdot \mathbb{1} = d^{t+1}$$
$$\forall i, j, t : P_{ij}^t \geq 0.$$

Note that the d^t's are automatically distributions by the above constraints and the fact that d^1 is. This problem

is equivalent to the following problem. First compute the lowest cost, C^*, for transporting flow from a pixel in the first frame to one in the last frame by finding shortest paths through the graph given by connecting all pixels of one frame with all pixels in the next frame and costs $\{C^t\}_{t=1}^{N-1}$. Using C^*, we are reduced to the classic EMD problem (5). From its solutions we can then construct all the P^t's and d^t's by following the shortest paths computed for finding C^*. As in (Cuturi, 2013), we could speed up the computation time of this equivalent problem by adding an entropic regularisation term. But this might be insufficient to train the neural network well as there is still the discrete computation of shortest paths involved. As discussed above shortest paths behave "jumpy" in the costs and are thus hard to differentiate through. We can try to avoid this by regularising even further. Instead of one global entropic term we could add entropic regularisers for each P^t in (6). To ease notation, we also substract the constant $\frac{N-1}{\lambda} = \frac{1}{\lambda} \sum_{t,i,j} P_{ij}^t$. The resulting Lagrangian is:

$$L\left(\{P^t, \alpha^t, \beta^t\}_{t=1}^{N-1}\right) =$$
$$\sum_{t,i,j} P_{ij}^t C_{ij}^t + \frac{1}{\lambda} \left(P_{ij}^t \log(P_{ij}^t) - 1\right) +$$
$$\alpha^{tT}(P^t \cdot \mathbb{1} - d^{t+1}) + \beta^{tT}(P^{tT} \cdot \mathbb{1} - d^t).$$

Setting its derivatives by P_{ij}^t to zero yields:

$$P_{ij}^t = \exp\left(-\lambda \alpha_i^t\right) \exp\left(-\lambda M_{ij}^t\right) \exp\left(-\lambda \beta_j^t\right).$$

In other words, the transport matrices again have the form

$$P^t = D(u^t) \cdot K^t \cdot D(v^t)$$

for $K^t = \exp\left(-\lambda M^t\right)$, $u^t = \exp\left(-\lambda \alpha^t\right)$, $v^t = \exp\left(-\lambda \beta^t\right)$ elementwise.[2] If the distributions d^t were all fixed, we could again use the simple iterations of the Sinkhorn algorithm (Sinkhorn, 1967) to find the P^t. They come from the fact that the row and column sums of P^t must be d^{t+1} and d^t, respectively, and are:

$$u^t = d^{t+1} \oslash K^t v^t \text{ and } v^t = d^t \oslash K^{tT} u^t, \qquad (7)$$

where $\oslash$ ($\otimes$) means elementwise division (multiplication). The zeros in d^1 and d^N can be handled as in (Cuturi, 2013) Algorithm 1. Since $d^2, \ldots, d^{T-1}$ are also optimisation variables, we have to update them too, for instance via:

$$d^{t+1} = u^t \otimes K^t v^t \text{ and } d^t = v^t \otimes K^{tT} u^t. \qquad (8)$$

So the complete solution algorithm would be to initialise $d^2, \ldots d^{N-1}$ and $\{u^t, v^t\}_{t=1}^{N-1}$ for instance as uniform distribution over the whole frame and as vectors of ones, respectively. Then, we would alternate between the iterations

---

[2]The modification of Sinkhorn's algorithm which we propose later automatically ensures the non-negativity of the P^t's although we have not explicitly added it to the Lagrangian.

in (7) and (8). This leads to many but very simple updates. Despite this not being a true instance of the Sinkhorn algorithm, we hope that the process converges fast. After some stopping criterion is reached, we could compute a cross entropy loss between the $\{d^t\}_{t=2}^{T-1}$ and distributions coming from the true position of the object. This loss could be differentiated by the d^t's and then by unrolling the iterations of (7) and (8) also by the costs, so that we can train the neural network. Once the network is trained we can use this approach to track an object in a video, given its position in the first and last frame. We could also make this approach more similar to the one before by only fixing d^1 and computing the loss just using d^N.

5. Implementation

We have implemented the shortest paths approach.[3] For the neural network we drew inspiration from the architecture of the neural network in (Lee et al., 2017). The network is a convolutional network with a shallow U-form. A ReLu layer follows each convolutional layer. As described in (Lee et al., 2017), at first two 2D convolutions are applied. We use a kernel size equal to the window size of the graph. The 2D convolutions increase the channel number from 3 over 5 to 10. Padding is used to keep the original dimensions. The result is recorded, let us call it X, and a copy is 2D max pooled to decrease the dimensions of the frames by a factor of 2. Then, the video is 3D convoluted to introduce time dependency. The 3D convolutions keep the channel number constant. First, we use a kernel of dimensions (3, window size, window size). So, the convolution sees a square area of length window size in three consecutive frames. It should be noted, that this corresponds to information being convoluted that comes from an area of side length twice the window size as we have pooled before. We apply padding to keep the dimensions. Afterwards, we use a kernel of dimensions (2, window size, window size) with padding to keep the latter two dimensions. As a result, we reduce the time dimension by one. Then we upsample trilinearly keeping the time dimension but restoring width and height of the frames to the original values, let us call the result Y. Before adding the 2D information X and the 3D information Y, we convolute X in the time dimension with a kernel of size 2 to match up dimensions. In the last step, we apply a 2D convolution layer with a kernel of window size, which gives out window size×window size many channels. Here we do not apply a ReLu layer. We interpret the channels of the output as the costs for edges going out of a pixel.

The graph is built explicitly using the graph_tool package. Since we build a large graph, we chose this performant

---

[3]https://github.com/sdamrich/FlowTracker/tree/submission

Table 1. Overlap of prediction and ground truth

		Iterations		
		10	50	200
Window size	3	0.227	0.302	0.411
	5	0.094	0.110	0.075
	9	0.000	0.000	0.000
	13	0.000	0.000	0.000

package.[4] Finally, we used a Mean Squared Error between the computed and the updated costs as loss for the neural network.

6. Experiments

6.1. Datasets

The videos we used belong to the visual object tracking challenge dataset of the year 2016. To be precise, we utilized the VOT2016 and VOT2016 Ground Truth datasets available under the following links http://www.votchallenge.net/vot2016/dataset.html. We cropped the video "ball1" to an area around the object of interest and shrunk the result to 64×64 pixels per frame to avoid long computations and size problems.

6.2. Results

We have tested the algorithm for window sizes 3, 5, 9 and 13 and training iterations for the network of 10, 50 and 200 on the first 5 frames of the cropped and shrunk video "ball1". The resulting predictions are shown in figure 1. The ball moves upwards between the first and last frame. The main result is that the algorithm tends to preserve the shape of the ball, but does not follow its movement, even though it was trained and tested on the same video. Therefore, we refrained from scaling up our experiments to several training and test videos.

We see that the anticipated over-merging problem from section 3.4.1 does not appear. The predictions in the last frame, though misplaced are rather dense and of the correct shape. We believe that this is because we use a convolutional network for computing the costs. When the network is told that some cost should be changed, it has to update its convolution filters which are applied in the same way to every area of the image. This way, the network avoids to only apply changes locally. As a result, no edges reduce their costs too much compared to the rest that there is the danger of over-merging of paths. In fact, the predictions are sparser for fewer iterations when the network only had little time to train. With more training the predictions become
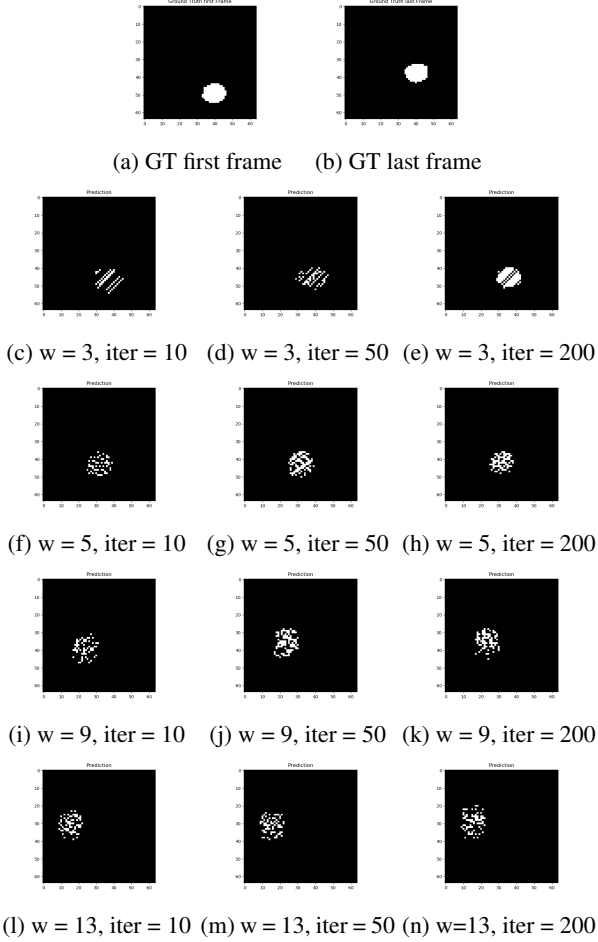
---

[4]https://graph-tool.skewed.de/performance

(a) GT first frame   (b) GT last frame

(c) w = 3, iter = 10 (d) w = 3, iter = 50 (e) w = 3, iter = 200

(f) w = 5, iter = 10 (g) w = 5, iter = 50 (h) w = 5, iter = 200

(i) w = 9, iter = 10 (j) w = 9, iter = 50 (k) w = 9, iter = 200

(l) w = 13, iter = 10 (m) w = 13, iter = 50 (n) w=13, iter = 200

Figure 1. Ground truth and predictions for $w \in \{3, 5, 9, 13\}$ and iter $\in \{10, 50, 200\}$

dense and the shape of the prediction tends looks more like a ball.

However, even with 200 iterations the random initialisation of the network seems to matter as we can see in figure 2, where the predictions for three different training periods with a window size of 5 and 200 training iterations are depicted. The ball is predicted to be at very different locations.

The MSE between the computed and updated costs does not change much in the course of the training, which is why we do not report it in detail. This is not surprising, as we always update every shortest path to the sink. So the only possibility for the MSE to decrease is if the updates along two paths cancel on a mutual part. In particular, even a perfect flow that exactly tracks the object, would result in a MSE of similar magnitude as one that entirely misses the object. One could avoid this by scaling the update by a factor which decreases with the overlap of the flow in the last frame and the ground truth or with the training iteration.

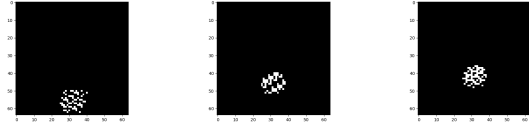A common effect seems to be that instead of just moving

Figure 2. Predictions of three different trainings with window size 5 and 200 training iterations.

upwards, the algorithm predicts the ball to also go to the left. This effect increases with the window size and with the number of iterations. We do not know why the object gets moved to the left by the algorithm and suspect an undetected programming error on our side.

It makes sense that the overall movement increases with the window size, simply because a larger window size allows for larger movement from frame to frame. The ball moves upwards so fast that it can only be properly tracked with window sizes 9 and 13. However, in these cases the algorithm predicts the ball even higher than it is in the ground truth ((j)-(n)). This might be because we did not scale down the updates for later iterations (or high overlap), so that there is no possibility for the training to converge. In most cases the prediction is as high and as far left as the window size allows. Only in the case of window size 13, the prediction could theoretically be even higher. The fact that the movement also slightly increases with the iterations might show that there is some burning in period.

One can see easily that the overlap of the predictions and the ground truth given by $\frac{\text{area of intersection of prediction and GT}}{\text{joint area of GT and prediction}}$ is rather small and in fact for most of our experiments the prediction misses the ground truth entirely. The exact values can be found in table 1. Better overlaps are achieved for small window size and high iterations. We believe that unfortunately, this is not because the smaller window size is generally better (they are even too small to theoretically track the object correctly) but since for small window sizes, the prediction has no chance to escape too much to the left.

7. Conclusion

There are several possible ways of using min-cost flows for tracking objects in videos. Such an optimisation problem looks at the whole video at once and does not propagate the position of the object iteratively through the video. This way information about the location of the object in different frames can be used in a unified manner. An issue with using MCFPs for object tracking is the large resulting problem size, which can make directly solving the linear problem or even differentiating through it prohibitively expensive. Another problem is the non-continuous change of an optimal flow with respect to the costs. Both the long computation time and the poor differentiability could potentially be mitigated by regularising the linear problem.

Thus, it would be interesting to pursue and implement the ideas of section 4 and to explore possbilities of transferring the ideas of (Schulter et al., 2017) to our case.

Acknowledgements

References

Amos, Brandon and Kolter, J. Zico. Optnet: Differentiable optimization as a layer in neural networks. *CoRR*, abs/1703.00443, 2017. URL http://arxiv.org/abs/1703.00443.

Berclaz, Jerome, Fleuret, Francois, Turetken, Engin, and Fua, Pascal. Multiple object tracking using k-shortest paths optimization. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1806–1819, 2011.

Cuturi, Marco. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Lee, Kisuk, Zung, Jonathan, Li, Peter, Jain, Viren, and Seung, H. Sebastian. Superhuman accuracy on the SNEMI3D connectomics challenge. *CoRR*, abs/1706.00120, 2017. URL http://arxiv.org/abs/1706.00120.

Lenz, Philip, Geiger, Andreas, and Urtasun, Raquel. Followme: Efficient online min-cost flow tracking with bounded memory and computation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4364–4372, 2015.

Ochs, Peter, Ranftl, René, Brox, Thomas, and Pock, Thomas. Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pp. 654–665. Springer, 2015.

Schulter, Samuel, Vernaza, Paul, Choi, Wongun, and Chandraker, Manmohan. Deep network flow for multi-object tracking. *arXiv preprint arXiv:1706.08482*, 2017.

Sinkhorn, Richard. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

Zhang, Li, Li, Yuan, and Nevatia, Ramakant. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.