Learning to Update for Object Tracking with Recurrent Meta-learner

Bi Li, Wenxuan Xie, Wenjun Zeng, *Fellow, IEEE,* and Wenyu Liu, *Senior Member, IEEE*

***Abstract*—Model update lies at the heart of object tracking. Generally, model update is formulated as an online learning problem where a target model is learned over the online training set. Our key innovation is to *formulate the model update problem in the meta-learning framework and learn the online learning algorithm itself using large numbers of offline videos*, i.e., *learning to update*. The learned updater takes as input the online training set and outputs an updated target model. As a first attempt, we design the learned updater based on recurrent neural networks (RNNs) and demonstrate its application in a template-based tracker and a correlation filter-based tracker. Our learned updater consistently improves the base trackers and runs faster than realtime on GPU while requiring small memory footprint during testing. Experiments on standard benchmarks demonstrate that our learned updater outperforms commonly used update baselines including the efficient exponential moving average (EMA)-based update and the well-designed stochastic gradient descent (SGD)-based update. Equipped with our learned updater, the template-based tracker achieves state-of-the-art performance among realtime trackers on GPU.**

***Index Terms*—model update, meta-learning, recurrent neural network, object tracking.**

I. Introduction

OBJECT tracking is a crucial task in computer vision that deals with the problem of localizing one arbitrary target object in a video, given only the target position in the first frame. Typically, bounding boxes are used for representing the target position. Arbitrary target object implies that a dedicated target model[1] is needed for each target during testing. This is typically accomplished through online learning where the online training set[2] is extracted from the test video and a target model is initialized and updated on the fly.

In this work, we tackle the problem of model update: after an initial target model is built using reliable supervision in the first frame, how to exploit information in subsequent frames and update the initial model along with tracking? Model update is challenging because of *unreliable and highly correlated online training set*. The only reliable supervision for building a target model is the information in the first frame. After that, the online training set is collected based on predicted target position, which is not always reliable. When small errors in the training samples accumulate, model update can cause the drifting problem. Moreover, the extracted online training set is highly correlated since most of the training samples are simply the translated and scaled version of a base sample. Correlated samples are easy to fit and do not help much with the generalization to hard samples.

Recent works [1], [2] have investigated the possibility of no model update at all, and achieved remarkable tracking performance. These approaches can be interpreted as learning an invariant and discriminative feature extractor such that the target remains stable in the feature space and is separable from the background. However, learning a representation that is both invariant and discriminative for a long time is intrinsically difficult, as with time evolves, features that once are discriminative may become irrelevant and vice versa. Consider that when a red car drives into a dark tunnel, the red color becomes irrelevant, although it is discriminative before entering the tunnel. Instead of striving to construct a perfect model at the first frame, model update tries to keep up with the current target appearance along with tracking by constantly incorporating the new target information, and therefore eases the burden of feature representation. Moreover, by gradually adapting to the current video context, the tracking problem can be considerably simplified [3]. In scenarios where target exhibits multi-modality, model update is indispensable.

Generally, model update can be formulated as an online learning problem with two stages. First, an online training set is collected along with tracking. Then, the target model is learned on the training set using algorithms like stochastic gradient descent (SGD). Existing update methods typically suffer from the problem of large training set and slow convergence thus being too slow for practical use. Moreover, due to unreliable training data, regularizations and rules are carefully designed based on expertise in the field to avoid model drifting.

In this work, we advocate the paradigm for object tracking that *eases the heavy burdens of online learning by offline learning*. Offline learning is performed before the actual tracking takes place and the learned model is shared among all test videos. Online learning, in contrast, is conducted during tracking and the learned model is specific to each test video. Our key innovation is to *formulate the model update problem in the meta-learning framework and* learn the online learning algorithm itself using large numbers of offline videos, i.e., *learning to update*. The offline-learned update method, which

B. Li and W. Liu are with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China. (e-mail: libi@hust.edu.cn; liuwy@hust.edu.cn)

W. Xie and W. Zeng are with the Microsoft Research Asia, Beijing 100080, China.(e-mail: wenxie@microsoft.com; wezeng@microsoft.com)

[1]In this work, we only consider scoring-based target model that outputs the confidence of an image patch being the target. Note there are also position-regression based target models that, given an image patch, directly regress the target position.

[2]The framework of learning to update contains an offline phase and an online phase. The offline phase is referred to as training the updater with offline videos, whereas the online phase refers to tracking a new sequence. The term "(online) training set" means the set of image patches collected for updating the target model (during the online phase).

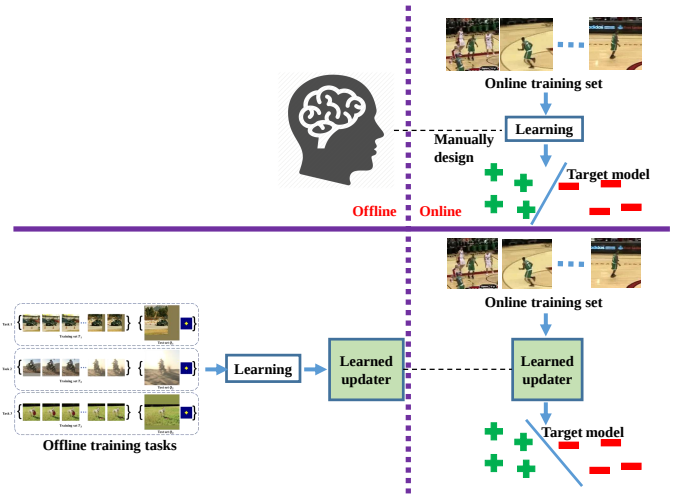


Fig. 1: A general introduction of learning to update. **Top**: Typically, a learning algorithm (e.g., SGD) is manually designed for online learning of the target model. **Bottom**: Contrarily, a learned updater is adopted for online learning, which is offline-trained using large numbers of videos. All figures are best viewed in color.

we call the learned updater, takes in the online training set and outputs the updated target model. Please refer to Fig. 1 for visual illustrations.

The benefit of learning to update is threefold: 1) After seeing all kinds of target variations in the offline training phase, the learned updater is able to capture *target variation patterns* among videos. These learned patterns are implicitly used during testing to avoid unlikely update (e.g., update to background) and thus can be seen as a form of regularization, which enables our learned updater to handle the unreliable online training set. 2) The learned updater is able to update the target model based on not only the online training set, but also rules learned from the offline dataset. Therefore, the learned updater is able to see beyond the highly correlated online training set and makes the updated model capable of generalizing to more challenging scenarios. 3) The learned patterns enable fast inference of the learned updater. As a result, our learned updater improves the performance of base trackers while running faster than realtime on GPU with a single forward pass of the neural network per frame.

In this paper, we formulate model update as a meta-learning problem (a.k.a learning to learn) [4], [5] and learn a model updater. Specifically, our learned updater is embodied as a RNN, which is well known for its ability to model sequential/temporal variations. Previous efforts to model target variations based on RNNs mostly fail to deliver satisfactory tracking performance due to inadequate offline training videos. In this work, we contribute several techniques to overcome data deficiencies and train RNNs effectively. With a properly trained updater, our tracker achieves state-of-the-art performance among realtime trackers.

As a first attempt of learning to update for object tracking, we demonstrate its application on two base trackers: a template-based tracker for its simplicity and a correlation filter-based tracker for its wide adoption. Our learned updater considerably improves the base trackers and outperforms relevant model update baselines including the exponential moving average (EMA)- and the SGD-based update method.

In summary, our contributions are threefold: 1) We propose a novel model update method for object tracking that (i) is formulated as a meta-learning problem, capable of learning target variation patterns and facilitating effective tracking, and (ii) runs faster than realtime (82 fps with SiamFC tracker and 70 fps with CFNet tracker) while requiring small memory footprint, thus being suitable for practical applications; 2) We propose several techniques to train our RNN-based updater effectively; 3) We validate our method in common object tracking benchmarks and show that it (i) consistently outperforms relevant model update baselines, and (ii) obtains state-of-the-art performance among realtime trackers.

II. Related Work

Handcrafted Update Methods. In general, target variation can be decomposed as short- and long-term variation. [6] proposes a probabilistic mixture model, which has a stable component to account for the long-term variation, a wandering component and a loss component for the short-term variation. Inspired by the Atkinson-Shiffrin Memory Model, [7] uses short- and long-term memory to handle target variations. Correlation filter and keypoint matching are employed for short- and long-term memory, respectively. Instead of using two separate components, we design a single component based on RNN and learn to process short- and long-term information in a data-driven manner.

One critical problem in model update is related to the stability-plasticity dilemma [8]. On one hand, model update should be stable to avoid the drifting problem where small errors accumulate and the model gets adapted to other objects. On the other hand, it also needs plasticity to effectively assimilate new information derived during tracking. [9] coined it the *template update problem*. They adopt a conservative update strategy which keeps the target model in the first frame (i.e., initial model), and updates the latest model only if its predicted locations are close to those of the initial model. Similar techniques are adopted in [10] which learns a ridge regression based on the first and the last target model. Inspired by this, we design an anchor loss that uses the first target model as an anchor point. We find it particularly useful in our learning based updater and will elaborate on it in Section IV.

Another strategy to handle the model drifting problem is being more careful about the derived training samples. Instead of making hard decisions about labeling training samples as target or background, [3] proposes a semi-supervised approach where only samples from the first frame are labeled and training samples from subsequent frames remain unlabeled. [11] proposes an occlusion detector and updates the model only if the occlusion level is low. [12] uses multiple-instance learning to update model with bags of samples, where positive bag contains at least one positive sample (without knowing which one), and negative bag contains only negative samples. [13] estimates the training sample qualities by optimizing the

target model loss with respect to both the target model and the sample qualities. We facilitate training information selection by adopting a gating mechanism in our learned updater.

A popular model update method for correlation filter-based trackers [14], [15] is exponential moving average (EMA), which performs linear interpolation from the newly trained model (using only training samples in the current frame) and the previous target model. This method is attractive because 1) the update process is highly efficient without iterative optimization and 2) training samples are processed on the fly without the need to be stored. However, it is unlikely that linear combination can capture all of the complex target variations. Our model update method outperforms EMA-based update while preserving all the practical benefits mentioned above.

Meta-learning. In this work, model update is formulated as a meta-learning problem. Essentially, meta-learning models a learning problem in two scales: learning for specific tasks, and learning for general patterns that rule specific tasks. In our case, updating target model for a specific target (e.g., an airplane) is a specific task. We aim to learn a model update method that is applicable to any specific tasks. [4], [5] learn to solve the optimization problem of neural networks based on RNNs. Their methods mainly focus on fast convergence and are not readily applicable for model update due to the unreliable training samples.

RNN-based trackers. RNNs are well known for their ability to model temporal variations. Given the importance of modeling temporal variations of target in object tracking, it is natural to consider taking advantage of RNNs. [16] is among the first to use RNN for object tracking, but has only shown to work on simple synthetic datasets. RATM [17] and HART [18] develop attention mechanisms based on RNNs and demonstrate success on natural image datasets KTH [19] and KITTI [20]. Re3 [21] models both appearance and motion variations using RNNs and achieves comparable results on several object tracking benchmarks [22]–[24]. However, Re3 [21] only models short-term variations and requires manual resetting of RNN states every 32 frames. RFL [25] proposes a filter generation method based on RNNs and resembles our method applied to the template-based tracker. Nevertheless, we tackle a more general model update problem and formulate our method in the meta-learning framework. RFL fails to deliver satisfactory tracking performance due to aggressive updating which is common in RNN-based trackers. By formulating the model update in the meta-learning framework, we focus on the generalization ability of the online-learned target model. With the proposed anchor loss, our approach outperforms RFL by a large margin. To the best of our knowledge, we propose the first RNN-based tracker that achieves state-of-the-art tracking performance among GPU-based realtime trackers.

III. Base Trackers and Base Update Methods

In this work, we focus on the update of *linear target model* due to its simplicity and wide adoption.

Basically, a target model is a scoring function that outputs the confidence of an input image patch being the target. Importantly, the target model should have parameters θ that is *updatable*. By model update, we mean updating θ such that it accommodates the target variations during tracking.

For a linear target model, the confidence c of an image patch with feature x is the inner product between θ and x, i.e., $c = \theta^T x$.

During tracking, a tracker gathers a training set T for updating the target model parameters θ. Since the dataset is gathered online, it is called the online training set. Let T_t be the online training set at time t, the target model is updated by an update function u, i.e., $\theta_t = u(T_t)$. This work is about *learning* an update function $u_\phi(\cdot)$ with parameters ϕ using large numbers of offline videos.

Before diving into the proposed learned updater, we introduce two base trackers with linear target model as well as two baseline model update methods in this section.

A. Template-Based Tracker: SiamFC

A template-based tracker simply uses the target feature (i.e., the feature of the target) as its target model. Intuitively, it means that the confidence of the test patch being the target is high when it is similar to the target feature and low otherwise. Due to the simplicity of the target model and the learning algorithm (i.e., an identity function), the performance of template-based tracker depends heavily on its features.

SiamFC [2] is a template-based tracker which achieves good performance with an offline learned feature extractor. The feature extractor is learned under a two branch structure with millions of image pairs sampled from videos. Each image pair contains a target patch and a test patch. The feature of the target patch is computed as the linear target model. The learning objective is that the confidence of a test patch containing the target is high while the confidence of background being low.

Interestingly, SiamFC does not have a model update module. In other words, given the target feature in the first frame $\bar{x}_1$ (we will use $\bar{x}$ for target feature and x for the feature of any image patch), the online training set contains only the target feature $T_t = \{\bar{x}_1\}$ and the learning algorithm simply takes the target feature as the target model $\theta_{t+1} = u(T_t) = \bar{x}_1$.

The detection process of SiamFC is equivalent to computing the similarity by sliding the target model θ over the search feature z and then outputting the position with the largest response. To facilitate fast detection, the feature extractor is designed to be fully convolutional and the similarity metric is simply inner product. Therefore, the detection process can be readily implemented via cross-correlation $\star$ (or convolution in the neural network literature):

$$p = \arg\max_p \theta \star z \ . \tag{1}$$

B. Correlation Filter-Based Tracker: CFNet

Correlation filters are one representative example of linear target model that have shown superior performance and gained a lot of popularity. The key factor behind the success of correlation filters is an efficient learning algorithm that is able to handle tens of thousands of circulant training samples. Correlation filter-based trackers have been improved in various

aspects since the seminal work of [14]. For simplicity, we use the basic formulation and follow the setup in CFNet [26].

In correlation filter-based tracker, given one base sample of the target feature $\bar{x}$, various virtual samples are obtained by cyclic shifts. The label of the base sample is 1 while the labels of the virtual samples follow a Gaussian function depending on the shifted distance. Given the samples and the corresponding labels, a ridge regression problem is solved efficiently in the Fourier domain making use of the property of circulant matrix [27]. In this work, we will simply use $\mathrm{CF}(\cdot)$ to denote the algorithm of learning the correlation filter from *one base sample with multiple feature channels*. Please refer to [14], [15] for more details about $\mathrm{CF}(\cdot)$.

The detection of correlation filter-based tracker is typically conducted in the Fourier domain. However, we convert the learned filter back to the spatial domain following CFNet. In this way, the detection process is the same as the SiamFC tracker. Moreover, it frees the learned updater from dealing with complex values.

C. Two Baseline Model Update Methods

For linear target model based trackers, there are two commonly used model update methods. We briefly introduce these two baseline methods in this subsection.

EMA-based model update. Let $g : \mathcal{X} \rightarrow \Theta$ be the algorithm of learning a linear target model using a *single* target feature. For template-based tracker, it is the identity function. For correlation filter-based tracker, it is $\mathrm{CF}(\cdot)$. Typically, the online training set contains the target feature at each frame, i.e., $T_t = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_t\}$. It is clear that $T_t = T_{t-1} \cup \{\bar{x}_t\}$. Given the online training set T_t, the updated target model via EMA is

$$\theta_{t+1} = u(T_t) \tag{2a}$$
$$= (1-\alpha)u(T_{t-1}) + \alpha g(\bar{x}_t) \tag{2b}$$
$$= (1-\alpha)\theta_t + \alpha\theta'_t \tag{2c}$$

where α is the learning rate that controls the rate of adaptation, $\theta'_t = g(\bar{x}_t)$ is the candidate target model at time t. Note that this update method only requires the last target model θ_t and the current target feature $\bar{x}_t$. Hence, there is no need to explicitly collect a large online training set.

EMA-based update is widely adopted in correlation filter-based trackers because it is easy to implement and efficient in terms of both memory consumption and computational complexity. Moreover, EMA-based update is the update rules of correlation filters derived for the multiple base samples with *single channel* case [14] and can be quite effective even for samples with multiple channels. However, in general, EMA-based update is ad hoc and only marginally improves the template-based tracker as shown in our experiments (Section V).

SGD-based model update. Besides EMA, another update method is to collect the online training set using samples from the search space of detection, i.e., $T_t = \{(z_1, y_1), (z_2, y_2), ..., (z_t, y_t)\}$ where z_t is the feature of the search image at frame t and y_t is the corresponding label map. The update process based on SGD is

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial l_{T_t^b}(\theta)}{\partial \theta} \tag{3}$$

where $l_{T_t^b}(\theta)$ is a differentiable objective function with variables θ over a batch of samples T_t^b from the online training set, and α is the learning rate.

Typically, a fixed size of online training set is maintained by dropping the earliest samples when the capacity is exceeded. However, it still contains thousands of samples. Moreover, we show only one iteration of SGD update for brevity while it generally requires dozens of iterations to take effects. These computational burdens of the SGD-based update method hinder its practical usage.

Fig. 2: Tasks for learning the model updater during offline training. Each task consists of a training set and a test set. The model updater should learn from the training set and generate a target model which is tested on the test set.

TABLE I: Correspondence between the meta-learning and learning to update. Best viewed by zooming in the electronic version.

meta-learning	**learn to update**	Explanation
meta-training	offline training	The process of learning a meta-learner (model updater)
meta-test	online training	The process of applying the meta-learner (model updater)
meta-learner	model updater	The model that takes as input an training set and outputs a learner (target model)
a task	a task	An example used for meta-training (offline training) the meta-learner (model updater), consisting of a training set and a test set
training set	training set	The dataset for learning a learner (target model)
test set	test set	The dataset for testing the learner (target model)
learner	target model	The model learned from the training set

IV. Learning to Update

In this section, we first formulate the model update problem in the meta-learning framework and then introduce the proposed model updater.

A. Model Update as Meta-Learning

Meta-learning is about learning from large numbers of tasks during meta-training to quickly learn a *new* task at meta-test time. Each task consists of a training set and a test set. The meta-learner (i.e. the model to be obtained via meta-training)

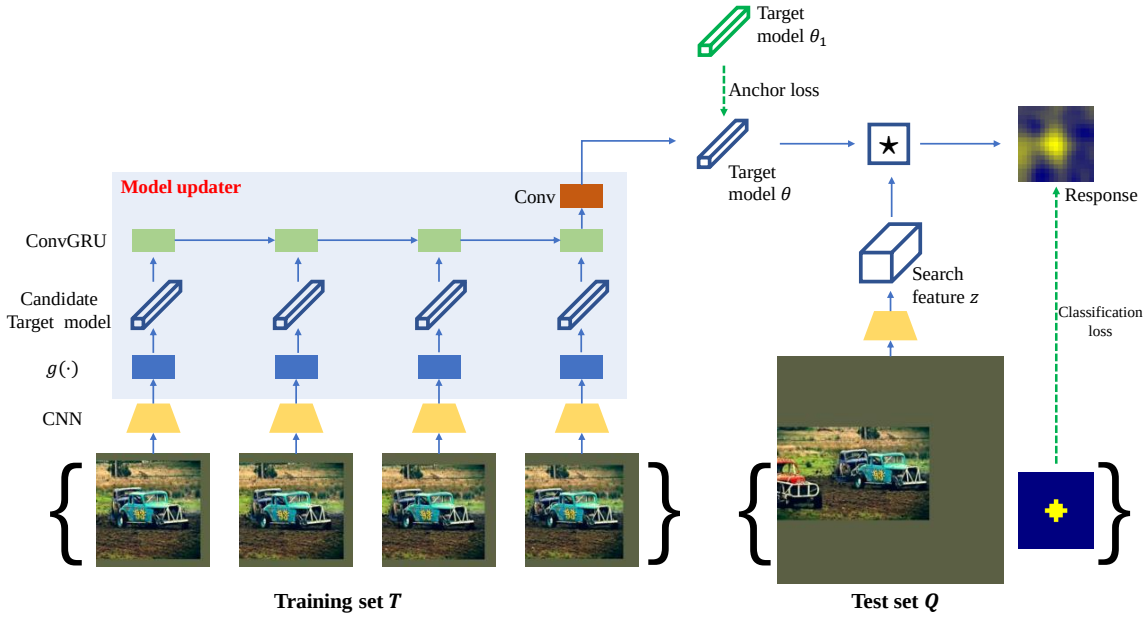


Fig. 3: The framework of learning to update during offline training. Given the training set T with image patches of a car, the target model θ is updated by the recurrent model updater. The target model is tested on the test set Q to obtain the classification loss. An anchor loss is also added to improve generalization. The model updater is learned by optimizing the anchor loss and the classification loss. During tracking, the model updater is fixed. An online training set is gathered along with tracking as T and the target model is updated by the model updater and applied to subsequent frames.

takes as input the training set of a task and outputs a learner which should perform well on the corresponding test set. The aim of meta-learning is to learn a meta-learner during meta-training such that the meta-learner can quickly learn a good learner at meta-test time, so meta-learning is also known as learning to learn.

The uniqueness of meta-learning is that, at meta-test time, the meta-learner should *quickly learn a new concept* using examples from the training set of the concept. An example is to learn a classifier to differentiate "Apple" and "Pear" based on examples of each category where these two categories never appear during meta-training. This resembles model update for object tracking since the tracker should quickly update the target model to accommodate an object that does not appear during offline training[3].

In the context of meta-learning for target model update, we aim to learn a meta-learner (model updater) from large numbers of offline videos during meta-training (offline training). Each task is to learn the learner (target model) from the training set to locate the target at test set. The correspondence between learning to update and meta-learning is summarized at Table I.

During offline training, given the training set T and the test set Q of a task constructed from the offline videos $(T, Q) \in \mathcal{V}$, the model updater computes target model $\theta = u_\phi(T)$. Let $l(\theta, Q)$ be the loss of the target model on the test set Q of a task. The model updater ϕ is learned by minimizing the following loss:

$$\mathcal{L}(\phi) = \sum_{(T,Q)\in\mathcal{V}} l(u_\phi(T), Q) \ . \tag{4}$$

[3]In fact, during offline training, the "human" category never appears whereas humans appear often in the tracking benchmarks.

B. The Training Set and the Test Set

We now describe how to construct the training set and the test set of a task for learning to update.

Given a video from the offline videos and the corresponding target positions (width, height, center position) at each frame, we first normalize the scale variations of targets by scaling the image with factor s such that $s(w + 2p) \times s(h + 2p) = A$ where w, h are the width and height of the target in the image, $p = \frac{w+h}{4}$ is the context margin and $A = 127 \times 127$ is the desired target size after scaling.

A subset of N image frames are sampled from the scaled images while keeping the temporal order. The first $N - 1$ frames are cropped at the center of the target, with size 127×127, and used as the training set of the target. The last frame is also cropped at the center of the target, with size 255×255, and used as the test set. Note that we use a larger image patch in the test set since both of our base trackers are translation equivalent and the 255×255 image can be seen as a set of 127×127 images. Please refer to Fig. 2 for several examples of training set and test set.

Cropped images are then embedded by the feature extractor. Finally, we have the training set $T = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_{N-1}\}$ where $\bar{x} \in \mathbb{R}^{m\times m\times d}$ is the feature of the target. The test set $Q = \{(z, y)\}$ where $z \in \mathbb{R}^{n\times n\times d}$ is the feature of the search image and $y \in \{-1, +1\}^{(n-m+1)\times(n-m+1)}$ is the

corresponding label map. Features have spatial size m or n and channel size d.

C. Instantiation of The Learned Updater

A model updater takes as input a training set $T = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_{N-1}\}$ and outputs the updated target model θ, i.e., $\theta = u(T)$. The design of the updater includes several preferable properties: 1) supporting training set with variable size; 2) incremental update, i.e., during tracking, the target model is updated based on existing values instead of learning from scratch; 3) memory and computational efficiency.

In this work, we propose a RNN-based updater that satisfies all these properties. Concretely, our updater follows a three-step procedure.

Step 1: Project from feature space to model space. We first project each target feature in the training set into the model space by $\theta' = g(\bar{x})$, where $\theta' \in \mathbb{R}^{m \times m \times d}$ is the candidate target model and $g(\cdot)$ is the algorithm of learning a linear target model using a single target feature. In particular, $g(\cdot)$ is the identity function for SiamFC and is the $\mathrm{CF}(\cdot)$ function followed by a center cropping function for CFNet.

Step 2: Aggregate target information. We use RNN to summarize the training set into a single tensor. For simplicity, gated recurrent unit (GRU) is adopted [28]. We find GRU achieves better performance than the Long-short term memory (LSTM [29]) in the ablation study. To preserve the spatial dimension, we extend the original GRU formulation to Convolutional GRU (ConvGRU) by replacing all matrix multiplications with convolutions.

Step 3: Generate target model. Given the last hidden state of the ConvGRU, one convolutional layer is used to generate the target model.

By adopting RNN, our updater is able to handle training set with variable size. Moreover, the target model is updated in an incremental manner. To make things clear, during tracking, denote $T_{t-1} = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_{t-1}\}$ as the online training set at time $t-1$. After model update, we obtain the hidden state h_{t-1}. At time t, the online training set is updated with a new example $\bar{x}_t$, i.e., $T_t = T_{t-1} \cup \{\bar{x}_t\}$. Since the first $t-1$ examples are unchanged, we can simply reuse h_{t-1} and get $h_t = \mathrm{ConvGRU}(h_{t-1}, \bar{x}_t)$. Moreover, with incremental update, we can avoid explicitly storing and manipulating a large online training set during tracking since our updater only needs h_{t-1} to generate h_t which saves a large amount of memory.

Until now, the updater is restricted to use $N-1$ target features as the training set and one search image feature as the test set. Note that given a sequence of $N-1$ target features, our updater can readily compute $N-1$ target models at each time step. Therefore, given a video with N images, we can construct various training sets with length $1, 2, ..., N-1$ and the computation for model update can be shared.

D. The Learning Objective

Given the updated target model θ, we need a "goodness" measurement of the target model which in turn indicates how good the updater is and thus enables optimization.

Classification Loss. Following the meta-learning framework, the updated model is evaluated on the test set $Q = \{(z, y)\}$. Using the normalized logistic loss for classification, we have

$$l_c(\theta; Q) = \frac{y(-\ln \sigma(\theta \star z)) + (1-y)(-\ln(1-\sigma(\theta \star z)))}{(n-m+1)\cdot(n-m+1)}. \tag{5}$$

Anchor Loss. At a first glance, it would seem that classification loss is all we need to train the updater. However, model update faces the intrinsic problem: *the stability-plasticity dilemma*, i.e., model update should be stable with respect to noise and flexible to assimilate new information. With only classification loss, since z is close to $\bar{x}_{N-1}$, the updater will adopt an aggressive update strategy and store new information brought by $\bar{x}_{N-1}$ as much as possible. The problem is that $\bar{x}_{N-1}$ is not always reliable during tracking and thus the updater trained with only classification loss is prone to small errors.

One effective method that is validated by the literature is to use the target model at the first frame as an anchor point [9], [10]. Given inadequate training data, such an anchor point is hard to learn without regularization. Therefore, we design an anchor loss, which penalizes the updater when the updated target model drifts away from the initial target model:

$$l_a(\theta; \theta_1) = \frac{1}{m \cdot m \cdot d} ||\theta - \theta_1||_2^2 \ . \tag{6}$$

where the loss is normalized by the number of target model parameters $m \cdot m \cdot d$.

Total Loss. Classification loss and anchor loss are linearly combined for measuring a target model:

$$l(\theta; Q) = (1-\lambda) l_c(\theta; Q) + \lambda l_a(\theta; \theta_1) \ , \tag{7}$$

where λ is the combination factor. Successful learning of the updater should maintain a good balance between the classification loss and the anchor loss.

The total loss of the updater $u_\phi(\cdot)$ with learnable parameters ϕ is then the loss of the target model in the offline training set by inserting Eq. 7 into Eq. 4.

E. Practical Techniques for Effective Learning

Our learned updater collects target information based on RNNs, which are well known for modeling sequential/temporal variations. However, the problem of limited offline training videos has to be addressed before it unleashes the power. We describe several techniques based on the nature of model update that turn out to be effective.

Modeling long-term variation by truncated backpropagation. Typically, a subset of N frames are sampled from the original video for RNN training. For convenience, we define *maximum modeling length* of an algorithm to be the largest length of all sampled sequences during training, where the length of a sampled sequence stands for the distance counted by *#(frames) in the original video* between the first and the last sampled frame. Since videos have hundreds of frames to track, it is desirable for RNN-based models to learn long-term dependency with a large maximum modeling length. However, training with long sequences is computationally demanding

and may incur the vanishing gradient problem. To avoid such a problem, existing RNN-based trackers sample relatively few frames sparsely from the original video. For example, [25] samples training sequences with 10 frames and large frame interval (30 frames on average). Such a sparse sampling strategy, however, enlarges the target variations between sampled frames, which is more difficult to learn. We conjecture that these trackers are disadvantaged by such limitations.

Contrarily, to train our learned updater, we sample training sequences with as many as 150 frames and small frame interval ($\leq$ 2 frames)[4]. To handle the aforementioned problem of training with long sequences, instead of backpropagating all the way to the first frame, we adopt truncated backpropagation. This method processes training frames one timestep at a time, and every H timesteps, it runs backpropagation through time (BPTT) for H timesteps. H is called the *unroll length*.

Matching training and testing behavior by estimated target position. During offline training of the learned updater, training set T of the training video needs to be generated. In our case, the online training set contains the target features at each frame $T_t = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_t\}$. The extraction of the target features depends on the target position which can only be estimated during tracking. RNNs often take as input the groundtruth during training, which is known as teacher forcing. However, as noted in [30], this causes discrepancy between training and testing and hampers the performance of RNNs. In this work, we always use the target position during training that is inferred by the target model to keep in line with testing.

Reducing overfitting by interval update. As noted in [31], instead of updating the target model in every frame, it is beneficial to apply a sparser update scheme. We adopt this simple strategy by updating the target model every M frames. Note that *the hidden state of RNN is updated in every frame* though. The reason for the improved performance is that by updating after a certain timesteps, the learned updater can make more informed update decision, and therefore reduces the risk of overfitting to current training samples.

V. Experiments

A. Implementation Details

Training data. The feature extractor and learned updater are trained offline on the ILSVRC 2015 Object Detection from Video dataset (Imagenet VID) [32]. Imagenet VID contains 4417 videos and each video has about 2 object tracks on average, adding up to 9220 tracks. Tracks are annotated with bounding boxes in each frame and contain about 230 frames on average. This differs from another large-scale video dataset, namely Youtube-BB [33] where objects are annotated every 30 frames and each track has about 15 annotated frames. We find that small frame interval is important for training the learned updater, and thus, we use Imagenet VID instead of the much larger Youtube-BB. However, it is worth investigating effective ways to make use of Youtube-BB for object tracking.

Image sequences are sampled from tracks as training samples for the learned updater. We use bucketing [34] (i.e., an RNN training technique which batch together sequences of similar lengths for efficiency) to handle sequences of different length. Multiples of the RNN unroll length are used as the bucket sizes. For example, bucket sizes of 25, 50, ..., 125, 150 are used for fast experimentation where 25 is the RNN unroll length. For tracks that are longer than the largest bucket size (e.g., 150), we sample a portion of the tracks with small frame interval (e.g., 1 or 2). For short tracks, we lengthen these tracks by duplicating frames or simply drop these tracks according to probabilities that are proportional to the track length. Images are preprocessed according to its base tracker SiamFC [2] and CFNet [26]. Particularly, these two base trackers use the same preprocessing procedures to crop and resize images such that targets are at the image center and take up 127 x 127 pixels together with context.

Architecture. For template-based tracker, we use the same modified Alexnet architecture in [2] for the feature extractor. For correlation filter-based tracker, we use the 3 layer CNN feature extractor in [26], which is trained following the procedures in [2]. All of our experiments stack two convolutional GRU layers, where convolution operations have kernel size 3 with zero padding to preserve spatial dimension. For template-based tracker, each convolutional GRU layer has 192 units while for correlation filter-based tracker, each has 64 units[5]. One convolutional layer with kernel size 3 is used to generate the updated target model based on the hidden states, which takes as input the concatenated states of the two convolutional GRU layers and outputs corresponding target models. Dropout [35] and layer normalization [36] are added in each convolutional GRU layer to avoid overfitting.

Optimization. Learned updaters are trained over 60 epochs, each epoch consists of 8309 image sequences. Gradients are computed using mini-batches of size 8, which are used by the Adam optimizer [37]. Learning rate is fixed to be 1e-4. Weight decay is 5e-4.

Hardware and software specifications. The speed measurements of our trackers are performed on a computer with an Intel Core i7-5930K Haswell-E 6-Core 3.5GHz CPU and a GeForce GTX 1080 GPU. Our trackers are implemented in TensorFlow [38], which is compiled with CUDA 8.0 and cuDNN 6.0.

Postprocessing. We adopt the same strategy as our base trackers for penalizing large displacement and handling scale variations. Specifically, a cosine window is added to the response map to penalize the large displacement. For scale estimation, three search patches with different scales are extracted and the current scale is calculated by interpolating the newly predicted scale with a damping factor.

B. Benchmarks and Evaluation Protocols

OTB. The OTB benchmark contains three subdatasets: OTB-2013, OTB-50 and OTB-100, each of which consists of 51, 50 and 100 natural image sequences, respectively. The standard evaluation metric on OTB is the area under curve

[4]In this sense, the maximum modeling length of our method here is 300 frames.

[5]The number of hidden units are set by searching from 32 to 384 with step size 32. We empirically find that setting the number of hidden units close to the channel size of the target model performs well.

(AUC) of the threshold-success rate curve which represents the success rates at different thresholds. For each frame, the overlap (intersection over union) between the predicted target bounding box and groundtruth is computed. The success rate at a given threshold corresponds to the fraction of frames that has overlap no less than the given threshold.

VOT. The VOT benchmarks are a collection of tracking challenges held on a yearly basis starting from 2013. We use three recent benchmarks: VOT-2015, VOT-2016 and VOT-2017. Unlike OTB, which lets the tracker run until the end of the image sequence, VOT focuses on short-term tracking (no redetection is required) and resets the tracker once it drifts away from the target. The primary measure is the expected average overlap (EAO), which reflects the similar property as AUC. VOT-2017 also introduces a new "realtime challenge", where a tracker is constantly receiving images in realtime speed and if the tracker does not respond after a new frame becomes available, the last bounding box predicted by the tracker is reported for the current frame.

C. Ablation Study

We validate the effectiveness of various designs of our learned updater based on the template-based tracker. OTB-2013 is used for ablation study. All experiments use the same configurations except the components that are examined. Although larger unroll length typically gets better results, in the ablation study section, we use unroll length 25 for fast experimentation which is also the default configuration unless larger unroll length is needed. Moreover, only color images are adopted during offline training to prevent benchmark-specific choices[6]. All models are trained using our implementation including SiamFC. Results are summarized in Table II.

Anchor loss is crucial for successful learning. To overcome the stability-plasticity dilemma, we propose the anchor loss which penalizes large variations of the updated target model. To minimize the anchor loss, one straightforward strategy would be *no update at all*. However, besides the anchor loss, the learned updater is also constrained by the classification loss which encourages the update of the target model to keep up with target variations. It is of interest to investigate how the interplay between these two losses affects the performance of the learned updater.

Our learned updaters trained with different combination factors λ are shown in Table II(d). We also include the results of the no-update baseline (i.e., the setup in SiamFC [2]) for reference. When $\lambda = 0$ (i.e., no anchor loss), our learned updater is not constrained to generate target models that are consistent with the initial target model and can quickly drift away. The performance is even worse than not updating at all. As λ increases, the learned updater gets better and reaches the peak at 0.2. Further increasing the anchor loss weight diminishes the possible target model choices of the learned updater and the performance drops. Note that even without the classification loss (i.e., $\lambda = 1$), our learned updater outperforms the baseline with no update. The reason is that the learned updater is given the initial target model only once. After that, it constantly receives new target model information and the hidden states of RNN inevitably stores new information due to the soft store operations. It is this new information that helps tracking.

Learned updater is orthogonal to feature extractor. Our template-based tracker is based on [2], which aims to learn an invariant and discriminative feature extractor such that model update is not necessary. Two interesting questions are: 1) how much variation the Siamese network is able to learn, and 2) how the learned feature extractor affects our learned updater. We answer these questions from the perspective of training samples of feature extractor. Every training sample of the Siamese network contains two image patches from two frames of a video. These two patches are both centered on the target and at most K frames apart, and therefore the maximum modeling length is K according to the definition in Section IV-E. We investigate the effects of different K, and train an updater based on each feature extractor. Results are shown in Table II(e).

As can be seen, 1) Siamese network is able to capture the variations of target within 100 frames, but has difficulties learning beyond this limit. 2) Although the feature extractor is trained with the objective of invariance and does not require model update, our learned updater consistently improves the base trackers. Moreover, the improvement is *positively correlated* with the performance of the feature extractor.

Train longer, generalize better. Model update is a process that typically spans hundreds of frames. We investigate the effects of training with long image sequences (large maximum modeling length). Table II(f) shows the tracking results of the learned updater trained with different maximum modeling length and unroll lengths. By increasing the maximum modeling length from 100 to 300, the learned updater monotonically gets better results. Since modeling long term dependencies is still a challenge for RNN, the performance degenerates when it reaches 400. In conclusion, 1) large maximum modeling length helps the learned updater to generalize better if it is within the modeling capabilities of RNN; 2) large unroll length is still helpful under truncated backpropagation to model long term dependencies.

Moreover, it is worth mentioning that the performance of the Siamese network decays as the maximum modeling length is over 100 (as shown in Table II(e)). We have tried to increase the number of neurons in Siamese network, but it does not help. Contrarily, our learned updater can handle longer sequences (e.g., 300 frames). It can be inferred that it is difficult to extract invariant features to handle long-term target variation, whereas learning an updater to adapt the target model gradually is relatively easier.

Practical techniques are helpful. We demonstrate the effectiveness of the various practical techniques introduced in Section IV-E by removing one component at a time. Note that small unroll length is used by default instead of large unroll length for fast experimentation. The results are summarized in Table II(a). As shown in the table, every component has contributed to the final performance. Among

[6]OTB-2013 contains both color and grayscale videos. Therefore, models trained with both color and grayscale images typically perform better in this benchmark.

TABLE II: **Ablations** on the OTB-2013 dataset using template-based tracker. Only color images are adopted during offline training. RNNs are unrolled 25 steps by default.

(a) **Effective techniques:** Cross mark means the technique is removed. RNNs are unrolled 25 steps by default unless large unroll length (50) is adopted. Image sequence are sampled by default with small frame interval (≤ 2). By removing the technique, we use large frame interval (≥ 10).

interval update?					✗
estimated target position?				✗	
small frame interval?			✗		
large unroll length?		✗	✗	✗	✗
AUC (%)	64.4	63.6	63.0	62.6	61.7

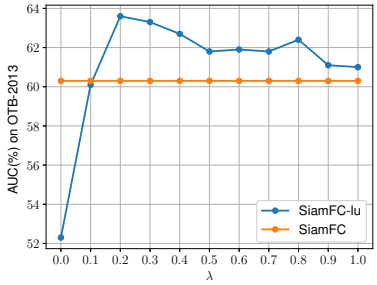
(b) **RNN configurations**: For example, ConvGRU-192x2 denotes stacking 2 layers of ConvGRU, each with hidden size 192.

configuration	AUC (%)
ConvGRU-192x1	62.1
ConvGRU-192x2	**63.6**
ConvGRU-192x3	63.3
ConvGRU-128x2	62.8
ConvGRU-192x2	**63.6**
ConvGRU-256x2	63.4
ConvLSTM-128x2	62.2
ConvLSTM-192x2	62.8
ConvLSTM-256x2	62.6

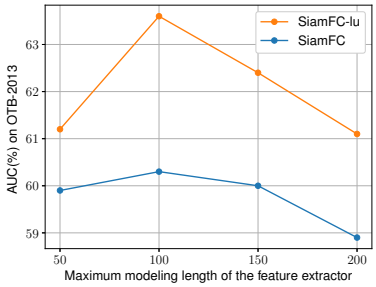
(c) **Joint training:** F: feature extractor, M: model updater, F + M: joint training of feature extractor and model updater.

	Description	AUC (%)
scheme 1	stage 1: F + M	51.5
scheme 2	stage 1: F stage 2: F + M	56.2
scheme 3	stage 1: F stage 2: M stage 3: F + M	63.1
Ours	stage 1: F stage 2: M	**63.6**

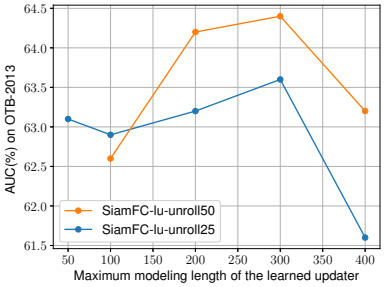
(d) **Anchor loss:** Tracker accuracy as the combination factor λ varies during updater training. "lu" is our learned updater.



(e) **Maximum modeling length of feature extractor:** Tracker accuracy as the maximum modeling length of the feature extractor varies during offline training, with and without the learned updater.



(f) **Maximum modeling length of model updater:** Tracker accuracy with varied maximum modeling length and unroll length of the model updater during offline training.



these techniques, interval update plays such an important role that the performance drops about 2% once removed. This is in accordance with the findings in [31].

Ablation on the RNN. Table II(b) shows the results of RNNs with different configurations of cell unit (ConvGRU or ConvLSTM), number of stacked RNN layers and the hidden state size. As can be seen, ConvGRU consistently outperforms ConvLSTM which may be related to the observation that the best performing cell unit is task-dependent [39]. We also observe that the AUC largely increases from 62.1% to 63.6% by stacking two layers of ConvGRU. The performance is not sensitive to the size of the hidden state as long as it is in a reasonable range (192˜256).

Joint training is difficult. The whole system (including feature extractor and model updater) can be trained jointly. We have tried several schemes for joint training, of which the results are summarized in Table II(c). Interestingly, it is best to train feature extractor and model updater separately - first train the feature extractor without model updater, and then fix the feature extractor and train the model updater. Scheme 3 jointly trains feature extractor and model updater after separately learning these two components. However, the performance still degrades. The reason is arguably that the ability of the feature extractor for modeling appearance invariance is hampered during joint training since the frame interval between selected images is small.

TABLE III: Comparisons with three representative baselines: no update, EMA-based update and SGD-based update. The AUC and EAO metric (higher is better) are reported for OTB and VOT, respectively. For OTB only, the feature extractors are trained with both color and grayscale images.

	OTB-2013	OTB-100	OTB-50	VOT-2015	VOT-2016	VOT-2017	Speed(FPS)
SiamFC-no-update	0.608	0.582	0.516	0.290	0.235	0.188	117
SiamFC-ema	0.618	0.597	0.538	0.286	0.259	0.216	91
SiamFC-sgd	0.644	0.614	0.563	0.306	0.278	0.248	13
SiamFC-lu (Ours)	**0.657**	**0.620**	**0.577**	**0.318**	**0.295**	**0.263**	82
CFNet-no-update	0.568	0.541	0.501	0.219	0.201	0.173	134
CFNet-ema	0.608	0.580	0.550	0.237	0.229	0.189	79
CFNet-sgd	0.613	0.590	0.555	0.235	0.212	0.182	9
CFNet-lu (Ours)	**0.621**	**0.599**	**0.565**	**0.242**	**0.230**	**0.208**	70

D. Baseline Comparisons

We consider three relevant baselines: no update, EMA-based update and SGD-based update. Please refer to Section III for an introduction of the EMA-based and SGD-based update.

- **No update**: the target model is initialized in the first frame and then remains fixed.
- **EMA-based update**: the last target model and the current candidate target model are linearly interpolated. The learning rate α is searched on OTB-2013 from 0.01 to 0.2 with step size 0.01.
- **SGD-based update**: we adopt the short-term update and long-term update following MDNet[7] [40]. Short-term update is triggered when the tracker has low confidence

[7]Unlike MDNet which updates the last 3 layers, we only update the linear target model, i.e., the last layer for meaningful comparison.

TABLE IV: Comparisons with state-of-the-art trackers.

(a) **OTB and VOT:** Trackers are split into two groups: realtime trackers and non-realtime trackers. The AUC and EAO metric (higher is better) are reported for OTB and VOT, respectively. Red and blue fonts indicate *1st* and *2nd* best performance of each group, respectively.

	OTB-2013	OTB-100	OTB-50	VOT-2015	VOT-2016	VOT-2017	Speed(FPS)
ECO	0.709	0.694	0.643	–	0.374	0.280	6
MDNet	0.708	0.678	0.645	0.38	–	–	1
LSART	0.677	0.672	–	–	0.324	0.323	1
CSRDCF	–	0.587	–	0.320	0.338	0.222	13
CREST	0.673	0.623	–	–	0.283	–	1
RFL	–	0.581	–	–	–	0.222	15
SiamFC-lu (Ours)	0.657	0.620	0.577	0.318	0.295	0.263	82
EAST	0.638	0.629	–	0.34	–	–	159
DSiam	0.656	–	–	0.293	–	–	25
CFNet-lu (Ours)	0.621	0.599	0.565	0.242	0.230	0.208	70
CFNet	0.610	0.589	0.538		–	–	79
SiamFC	0.608	0.582	0.516	0.290	0.235	0.188	117

(b) **VOT-2017 realtime challenge:** Results of top-performing trackers and our trackers on the VOT-2017 realtime and non-realtime (also called baseline) challenge. The EAO metric (higher is better) is reported.

challenge	**SiamFC-lu (Ours)**	CSRDCF++	**CFNet-lu (Ours)**	SiamFC	ECO-HC	LSART	CFCF	ECO
realtime	0.258	0.212	0.200	0.182	0.177	0.055	0.059	0.078
non-realtime	0.263	0.229	0.208	0.188	0.238	0.323	0.286	0.280

while long-term update is conducted every 10 frames. The hyperparameters of the SGD-based update (e.g., online training set size 1000, batch size 8, learning rate 10, number of iterations, 500 for long-term, 200 for short-term, etc.) are searched on OTB-2013.

For fair comparison, we retrain the updater using publicly available feature extractors (SiamFC and CFNet are open-sourced) with configurations validated in the ablation study. As it turns out, the publicly available feature extractor of SiamFC trained with both color and gray images improves the tracker performance from 0.644 to 0.657 on OTB-2013. This is in line with our observations that the performance of a tracker equipped with our learned updater is positively correlated to that of the feature extractor. The results are summarized in Table III. Qualitative comparisons are presented in Fig. 4.

Observations: 1) Our learned updater significantly outperforms the no update and EMA update baselines. 2) A somewhat surprising yet encouraging result is that our learned updater achieves better performance than the heavily designed and tuned SGD update method. 3) It is worth noting that for correlation filter-based tracker, EMA update is still a strong baseline. 4) Noticeably, the improvement on SiamFC is consistently larger than that on CFNet. As noted in [26], the $\mathrm{CF}(\cdot)$ may impose priors that become overly restrictive when enough modeling capacity and data are available. 5) Our learned updater outperforms the SGD-based update method while enjoying comparable efficiency as the EMA-based update method.

E. State-of-the-art Comparison

We compare trackers equipped with our learned updaters against 10 state-of-the-art trackers: ECO [31], MDNet [40], LSART [41], CSRDCF [42], CREST [43], RFL [25], EAST [44], DSiam [10], CFNet [26] and SiamFC [2]. The results are summarized in Table IV(a).

Observations: 1) SiamFC-lu achieves state-of-the-art performance among realtime trackers. 2) SiamFC-lu outperforms DSiam and RFL, which also focus on improving the update mechanism of SiamFC.

To evaluate the practicability of different tracking methods, VOT-2017 introduces a new "realtime challenge" that only allows the tracker to respond in realtime; otherwise, the last predicted target position will be used for the current frame. We compare our method with the state-of-the-art trackers in this setting: CSRDCF++ (a C++ implementation of the CSRDCF [42] tracker), SiamFC [2], ECO-HC (a lightweight version of ECO [31] that uses HOG feature), LSART [41] and CFCF [45]. The results are shown in Table IV(b). It can be observed that our approach achieves state-of-the-art results in the realtime setting.

VI. Discussion

One interesting question would be why learning to update actually works? We try to answer this question in three different perspectives. 1) From the high-level perspective, fast model update is viable because videos are intrinsically structured (e.g., temporal dependencies between target features, target variation patterns). Our learned updater captures these structures in a data driven manner. 2) As for the functionality, our learned updater can be seen as a learnable extension of the EMA-based update. The difference is that, instead of linearly interpolating the last target model and the current candidate target model, we adopt the gating mechanism. As a result, the learned updater inherits the efficiency of EMA-based update and the effectiveness of learning based method. 3) Empirically speaking, as shown in Fig. 4, after being trained under the classification loss and anchor loss, our learned updater is able to reliably absorb target variations while resisting the distractors.

As a first attempt, however, there are still many interesting problems left uninvestigated. One issue of RNN-based updater is that, the convolutional GRU requires large amount of GPU memory during offline training and thus being difficult to apply to target models with large numbers of parameters. How to scale to large target models would be an interesting research problem. Moreover, in this work, only linear target models are considered, how to extend to the non-linear cases such as MDNet is valuable future work.

VII. Conclusion

We propose a learning based framework to tackle the problem of model update during tracking. As a first attempt, only the update of linear target model is considered. The learned updater is parameterized based on RNN and successfully learned with several techniques proposed in this work. Our learned updater outperforms two common model update baselines including the efficient EMA-based update and the well-designed SGD-based update. After offline training, our learned updater can run efficiently during testing; therefore, our learned updater is able to consistently improve the base trackers without sacrificing the speed. Notably, the SiamFC tracker has been improved by nearly 40% in terms of the EAO on VOT-2017 while running at the speed of 82fps, which achieves state-of-the-art performance among realtime counterparts. In the future, we plan to extend the learning to update paradigm to non-linear target models.

Fig. 4: Qualitative results of our learned updater compared with common model update baselines. **Bolt**: Our learned updater correctly adapts to the target while others are attracted by distractors. SGD-based update method adapts to part of the target instead. **CarScale**: Other methods keep tracking part of the target since only the front of the car is shown in the first frame. Contrarily, our learned updater gradually adapts to the whole target including both the front and the tail of the car. **Matrix**: our learned updater is able to perform equally well compared with the SGD-based update in this challenging sequence. **Suv, Woman**: While all other methods fail, our learned updater still successfully tracks the target.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (grant No. 61733007, 61572207).

The authors would like to thank Chong Luo and Anfeng He for helpful discussions.

REFERENCES

[1] R. Tao, E. Gavves, and A. W. Smeulders, "Siamese instance search for tracking," in *CVPR*, 2016, pp. 1420–1429.

[2] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *ECCVw*, 2016, pp. 850–865.

[3] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised on-line boosting for robust tracking," in *ECCV*, 2008, pp. 234–247.

[4] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *NIPS*, 2016.

[5] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.

[6] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi, "Robust online appearance models for visual tracking," *TPAMI*, vol. 25, pp. 1296–1311, 2001.

[7] Z. Hong, Z. Chen, C. Wang, X. Mei, D. V. Prokhorov, and D. Tao, "Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking," in *CVPR*, 2015, pp. 749–758.

[8] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, vol. 11, pp. 23–63, 1987.

[9] L. Matthews, T. Ishikawa, and S. Baker, "The template update problem," *TPAMI*, vol. 26, no. 6, pp. 810–815, 2004.

[10] Q. Guo, W. Feng, C. Zhou, R. Huang, L. Wan, and S. Wang, "Learning dynamic siamese network for visual object tracking," in *ICCV*, 2017, pp. 1–9.

[11] S. Kwak, W. Nam, B. Han, and J. H. Han, "Learning occlusion with likelihoods for visual tracking," in *ICCV*. IEEE, 2011, pp. 1551–1558.

[12] B. Babenko, M.-H. Yang, and S. J. Belongie, "Robust object tracking with online multiple instance learning," *TPAMI*, vol. 33, pp. 1619–1632, 2011.

[13] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Adaptive decontamination of the training set: A unified formulation for discriminative visual tracking," in *CVPR*, 2016, pp. 1430–1438.

[14] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *CVPR*, 2010, pp. 2544–2550.

[15] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *TPAMI*, vol. 37, no. 3, pp. 583–596, 2015.

[16] Q. Gan, Q. Guo, Z. Zhang, and K. Cho, "First step toward model-free, anonymous object tracking with recurrent neural networks," *arXiv preprint arXiv:1511.06425*, 2015.

[17] S. E. Kahou, V. Michalski, and R. Memisevic, "Ratm: recurrent attentive tracking model," in *CVPRw*, 2017.

[18] A. R. Kosiorek, A. Bewley, and I. Posner, "Hierarchical attentive recurrent tracking," in *NIPS*, 2017.

[19] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *ICPR*, 2004.

[20] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *I. J. Robotics Res.*, vol. 32, pp. 1231–1237, 2013.

[21] D. Gordon, A. Farhadi, and D. Fox, "Re3: Real-time recurrent regression networks for object tracking," *arXiv preprint arXiv:1705.06368*, 2017.

[22] M. Kristan, R. P. Pflugfelder, A. Leonardis, J. Matas, and L. Cehovin, "The visual object tracking vot2014 challenge results," in *ECCVw*, 2014, pp. 191–217.

[23] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. P. Pflugfelder, and L. Cehovin, "The visual object tracking vot2016 challenge results," in *ECCVw*, 2016, pp. 777–823.

[24] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *CVPR*, 2013, pp. 2411–2418.

[25] T. Yang and A. B. Chan, "Recurrent filter learning for visual tracking," in *ICCVw*, 2017, pp. 2010–2019.

[26] J. Valmadre, L. Bertinetto, J. F. Henriques, A. Vedaldi, and P. H. Torr, "End-to-end representation learning for correlation filter based tracking," in *CVPR*, 2017, pp. 2805–2813.

[27] R. M. Gray *et al.*, "Toeplitz and circulant matrices: A review," *Foundations and Trends® in Communications and Information Theory*, vol. 2, no. 3, pp. 155–239, 2006.

[28] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *NIPS*, 2015, pp. 1171–1179.

[31] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg, "Eco: Efficient convolution operators for tracking," in *CVPR*, 2017, pp. 6638–6646.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[33] E. Real, J. Shlens, S. Mazzocchi, X. Pan, and V. Vanhoucke, "Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video," in *CVPR*, 2017, pp. 5296–5305.

[34] V. Khomenko, O. Shyshkov, O. Radyvonenko, and K. Bokhan, "Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization," in *Data Stream Mining & Processing (DSMP), IEEE First International Conference on*. IEEE, 2016, pp. 100–103.

[35] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[36] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[37] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[39] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[40] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *CVPR*, 2016, pp. 4293–4302.

[41] C. Sun, H. Lu, and M.-H. Yang, "Learning spatial-aware regressions for visual tracking," *CVPR*, 2018.

[42] A. Lukezic, T. Vojír, L. C. Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," in *CVPR*, 2017, pp. 4847–4856.

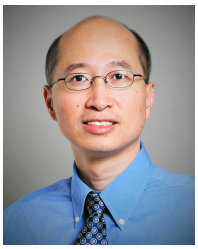
[43] Y. Song, C. Ma, L. Gong, J. Zhang, R. W. Lau, and M.-H. Yang, "Crest: Convolutional residual learning for visual tracking," in *ICCV*. IEEE, 2017, pp. 2574–2583.

[44] C. Huang, S. Lucey, and D. Ramanan, "Learning policies for adaptive tracking with deep feature cascades," in *ICCV*, 2017, pp. 105–114.

[45] E. Gundogdu and A. A. Alatan, "Good features to correlate for visual tracking," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2526–2540, 2018.

Bi Li received the B.Sc. degree from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014, and is currently a Ph.D. student at the media and communication lab, HUST, supervised by Prof. Wenyu Liu. His research interests include meta-learning, few-shot learning and object tracking.

Wenxuan Xie received the B.Sc. degree from Nanjing University, Nanjing, China, in 2010, and the Ph.D. degree from Peking University, Beijing, China, in 2015. He has been working as an associate researcher in Microsoft Research Asia since 2015. His research interests include computer vision and machine learning.



Wenjun (Kevin) Zeng (M'97-SM'03-F'12) is a Principal Research Manager and a member of the senior leadership team (SLT) of Microsoft Research Asia. He has been leading the video analytics research empowering the Microsoft Cognitive Services and Azure Media Analytics Services since 2014. He was with Univ. of Missouri (MU) from 2003 to 2016, most recently as a Full Professor. Prior to that, he had worked for PacketVideo Corp., Sharp Labs of America, Bell Labs, and Panasonic Technology. Wenjun has contributed significantly to the development of international standards (ISO MPEG, JPEG2000, and OMA). He received his B.E., M.S., and Ph.D. degrees from Tsinghua Univ., the Univ. of Notre Dame, and Princeton Univ., respectively. His current research interest includes mobile-cloud media computing, computer vision, social network/media analysis, and multimedia communications and security.

He was an Associate Editor-in-Chief of IEEE Multimedia Magazine, and was an AE of IEEE Trans. on Circuits & Systems for Video Technology (TCSVT), IEEE Trans. on Info. Forensics & Security, and IEEE Trans. on Multimedia (TMM). He was a Special Issue Guest Editor for the Proceedings of the IEEE, TMM, ACM TOMCCAP, TCSVT, and IEEE Communications Magazine. He was on the Steering Committee of IEEE Trans. on Mobile Computing and IEEE TMM. He served as the Steering Committee Chair of IEEE ICME in 2010 and 2011, and is serving or has served as the General Chair or TPC Chair for several IEEE conferences (e.g., ICME'2018, ICIP'2017). He was the recipient of several best paper awards. He is a Fellow of the IEEE.

Wenyu Liu (M'08-SM'15) received the B.S. degree in Computer Science from Tsinghua University, Beijing, China, in 1986, and the M.S. and Ph.D. degrees, both in Electronics and Information Engineering, from Huazhong University of Science & Technology (HUST), Wuhan, China, in 1991 and 2001, respectively. He is now a professor and associate dean of the School of Electronic Information and Communications, HUST. His current research areas include computer vision, multimedia, and machine learning. He is a senior member of IEEE.