Report

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

How to Train your DeepMOT

SMAI Course Project (Monsoon 2022)

Team 48

Naimeesh Narayan Tiwari (2020101074)

Soveet Kumar Nayak (2020101086)

Dhruv Hirpara (2020102029)

Atharva Joshi (2020111010)

2nd December, 2022

Git Repo

Abstract

We have implemented a Deep Hungarian Network as our differentiable framework which we can use with any tracktor using the Hungarian algorithm for Multi-object Tracking. The DHN approximates the Hungarian Algorithm and provides a soft approximation of the optimal prediction-to-ground-truth assignment. The proposed approximation is based on a bi-directional recurrent neural network (Bi-RNN) that computes the (soft) assignment matrix based on the prediction-to-ground-truth distance matrix. We pose the DHN training as a 2D binary classification task using the binary focal loss.

Keywords: Deep Hungarian Network, Hungarian (Munkres) Algorithm, Binary Focal Loss, Bi-RNN

Contents

1. Introduction

We are seeing much work being done on MOT, which exploits the power of deep learning. However, the current methods train individual parts of the MOT pipeline using loss functions that aren't directly related to the MOT evaluation measures. Loss functions that actually resemble the standard tracking evaluation measures are the need but the problem arises in **computing the optimal matching between the predicted object tracks and the ground-truth objects**. In the implementation, we have used the new differentiable framework for the training of MOTs from the paper. We combine the **CLEAR-MOT** evaluation measures from **"Keni Bernardin and Rainer Stiefelhagen - Evaluating multiple object tracking performance: The clear mot metrics"** with the novel loss function, which comes out to be suitable for end-to-end training of the MOTs.

1.1 Objective

The objective of any MOT method is to predict tracks in a video sequence. The objective of this method is to propose an end-to-end MOT training framework based on a differentiable approximation of HA and CLEAR-MOT metrics. The paper makes the following contributions:

- proposes novel loss functions that are directly inspired by standard MOT evaluation measures [6] for end-to-end training of multi-object trackers.
- proposes a new network module – Deep Hungarian Net – that learns to match predicted tracks to ground-truth objects in a differentiable manner.
- demonstrates improvements over the baseline and establishes a new state-of-the-art result on MOTChallenge benchmark datasets.

1.2 Background

The majority of existing methods for pedestrian tracking follow the tracking-by-detection paradigm and mainly focus on the association of detector responses over time. A significant amount of research investigated combinatorial optimization techniques for this challenging data association problem.

Tracking as Discrete Optimization

By using frame-to-frame bi-partite matching between tracks and detections, these approaches may be executed online and start by doing object detection in each image and associating detections over time.

Alternatively, tracking can be posed as a maximum-a-posteriori (MAP) estimation problem by seeking an optimal set of tracks as a conditional distribution of sequential track states. Several methods perform inference using conditional random fields (CRFs), Markov chain Monte Carlo (MCMC) or a variational expectation maximization. These techniques often leverage motion models, optical flow-based descriptors, or hand-crafted descriptors for the appearance model as association signals. As a result, only a limited number of parameters can normally be learned using grid/random search or tree of Parzen window estimators.

Deep Multi-Object Tracking

Several existing methods train parts of their tracking methods using losses, not directly related to tracking evaluation measures. Kim et al. leverages pre-learned CNN features or a bilinear LSTM to learn the longterm appearance model.

Xiang et al. learn track birth/death/association policy by modeling them as Markov Decision Processes (MDP). As the standard evaluation measures are not differentiable, they learn the policy by reinforcement learning.

In order to learn the parameters of linear cost association functions for multi-object trackers based on network flow optimization, Wang et al. present a framework. They use a structured SVM to train the parameters. They create a loss function that is similar to MOTA, with the intra-frame loss penalising false positives (FP) and missed targets and the inter-frame component of the loss penalising false associations, ID switches, and missed associations. However, given the suggested min-cost flow structure, their loss is not differentiable.

Another paper (Chu et al.) propose an end-to-end training framework that jointly learns feature, affinity and multi-dimensional assignment. However, their losses are not directly based on MOTA and MOTP metrics.

Schulter et al. parameterize (arbitrary) cost functions with neural networks and train them end-to-end by optimizing them with respect to the min-flow training objective.

Bergmann et al. propose a tracking-by-regression approach to MOT. Using a smooth L1 loss for the bounding box regressor, the algorithm is trained for the object detection problem. This approach is completely trainable, with the exception of the track birth and death management.

2. Data

We have used the MOTChallenge benchmark dataset from **Anton Milan, Laura. Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler: MOT16: A benchmark for multi-object tracking** and **Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler: MOTChallenge 2015: Towards a benchmark for multi-target tracking.**

3. Conceptual Discussion

The objective of any MOT method is to predict tracks in a video sequence. Each track X^{i} is associated with an identity 'i' , and consists of L_i image bounding boxes $x^{\mathrm{i}}{}_{t_l}$ belongs to R^4 (2D location and size), l = 1...,L_i.
At evaluation time, t, the N_t predicted bounding boxes must be compared to the M_t

ground-truth objects given by ${y_t}^{jM_t}$.

3.1 Hungarian Algorithm

To understand the full implementation and the need to bring DHN, we need to first understand the Hungarian (Munkres) Algorithm.

$$A^* = \mathrm{argmin}_{A\epsilon\{0,1\}^{NXM}} \sum_{n,m} d_{nm}a_{nm}, \text{ s.t } \sum_{m} a_{nm} \leq 1, \forall n; \sum_{n} a_{nm} \leq 1; \sum_{m,n} a_{nm} = \min\{N, M\}$$

where D is calculated using bi-partite matching.
By solving this integer program, we obtain a mutually consistent association between ground-truth objects and track predictions. The constraints ensure that all rows and columns of the assignment should sum to 1, thus avoiding multiple assignments between the two sets.

3.2 DeepMOT

For MOT evaluation, the general **two step strategy** is:
STEP 1: Compute CLEAR-MOT tracking evaluation measures is to perform bi-partite matching between the sets of ground-truth objects and of predicted tracks.
STEP 2: Count TP, FN, FP, IDS required to get MOTA and MOTP.
We follow the same two step strategy here but with a differentiable loss function:
STEP 1: Soft matching done using a differentiable function parameterized as a Deep Neural Network.
STEP 2: Once we establish the matching, we design a loss as a combination of differentiable functions of the (soft) assignment matrix (Ã) and the distance matrix (D) approximating the CLEAR-MOT measures.

3.3 Deep Hungarian Net

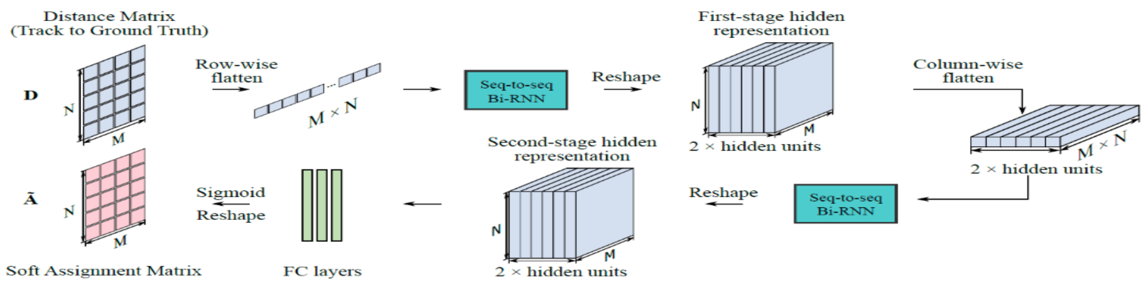

Figure 1: Structure of Deep Hungarian Net

Replacing A* with a soft approximation, DHN produces a proxy Ã that is differentiable w.r.t. D. We model DHN by a neural network, $\tilde{\mathbf{A}} = \mathbf{g(D,\ d)}$, with parameters d.

The DHN mapping must satisfy:
1. The output Ã must be a good approximation to the optimal assignment matrix A*.
2. This approximation must be differentiable w.r.t. D.
3. Both input and output matrix are of equal, but varying size.
4. g must take global decisions as the HA does.

We will achieve these by:
1. Setting appropriate loss function when training DHN.
2. Designing DHN as a composite of differentiable functions
3. Making every neutron have a receptive field equivalent to the entire input.
4. We have Bi-RNNs that can guarantee the above even in large assignment problems.

The sub-components of our DHN are:

3.3.1 Bi-RNNs

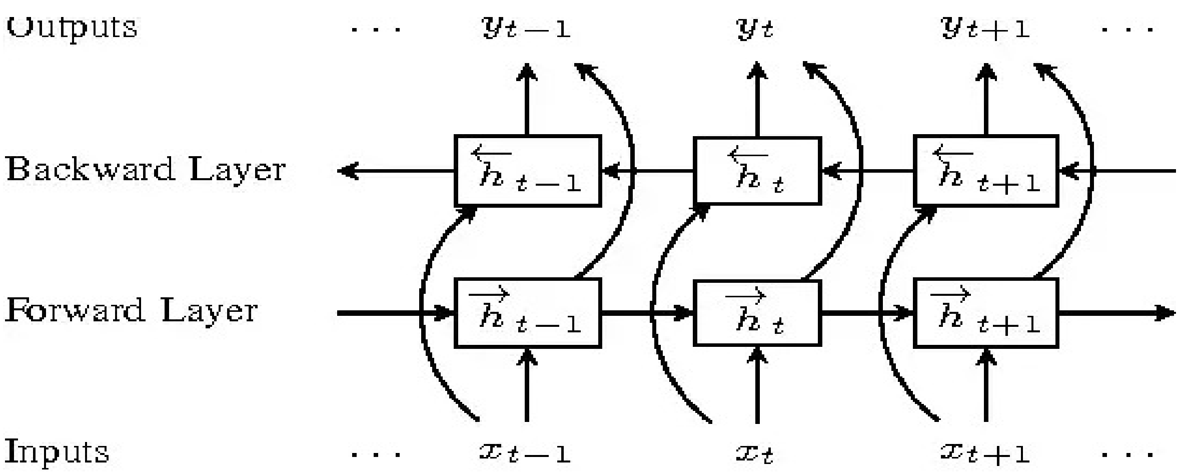

Figure 2: Bi-RNN

In bidirectional RNNs, the hidden state for each time step is simultaneously determined by the data prior to and after the current time step. Bidirectional RNNs are mostly useful for sequence encoding and the estimation of observations given bidirectional context. Bidirectional RNN (BRNN) duplicates the RNN processing chain so that inputs are processed in both forward and reverse time order. This allows a BRNN to look at future context as well.

3.3.2 Distance Matrix Computation

The most common metric for measuring the similarity between two bounding boxes is the Intersection-over-Union (IoU). Now input D can be any differentiable distance matrix, but if we keep it just as the IoU, then in case of no intersection between bounding boxes, i.e. true negative, the distance will come out to be 1-IoU which will be a constant 1, which will result in a gradient loss of 0 giving no information to be back-propagated. Hence, we define our distance as the average of the Euclidean center-point distance and the Jaccard distance J:

$$d_{nm} = \frac{f(x^n, y^m) + \mathcal{J}(x^n, y^m)}{2}$$

where f is the normalized Euclidean distance w.r.t. image size:

$$f(x^n, y^m) = \frac{||c(x^n) - (x^m)||_2}{\sqrt{H^2 + W^2}}$$

Hence, due to normalization, all entries in D are in the range [0,1].

3.3.3 Explanation of the Overall Structure

The original HA performs sequential row-wise and column-wise reductions hence mimicking it we perform row-wise and column-wise flattening of D. We perform it sequentially by first flattening it row-wise then inputting it to the Bi-RNN that gives a hidden representation of size NxMx2h where h is the length of the hidden representation. After flattening it row-wise we perform column-wise flattening and do the same. Then using a sigmoid we produce the optimal soft-assignment matrix of NxM dimensions Ã $[0, 1]^{\mathrm{N} \times \mathrm{M}}$.

3.3.4 DHN Training

For training the DHN, we form a dataset with D and A*, separated into 114,483 matrices for training and 17,880 for matrices testing. Using MOTChallenge datasets we generate D using ground-truth bounding boxes and public detections, then using HA we generate assignment matrices. We pose the DHN training as a 2D binary classification task using the focal loss.
The class imbalance is compensated by weigthing the zero-class using $w_0 = n_1/(n_0+n_1)$. We weight the one-class by $w_1 = 1 - w_0$. We evaluate the performance of DHN by computing the weighted accuracy WA.

$$WA = \frac{w_1 n_1^* + w_0 n_0^*}{w_1 n_1 + w_0 n_0}$$

Since the output of the DHN is between 0 and 1 we have kept the Threshold as 0.5.
Once the DHN is trained, We keep the weights fixed.
Now we can use this DHN for approximating Hungarian Algorithm. Hence any MOT using HA can directly use this module instead of HA for getting differential losses.

4. Analysis and Results

After training for over 11000 iterations for about 1 hour, we finally get these values for the following parameters as given in the plots:

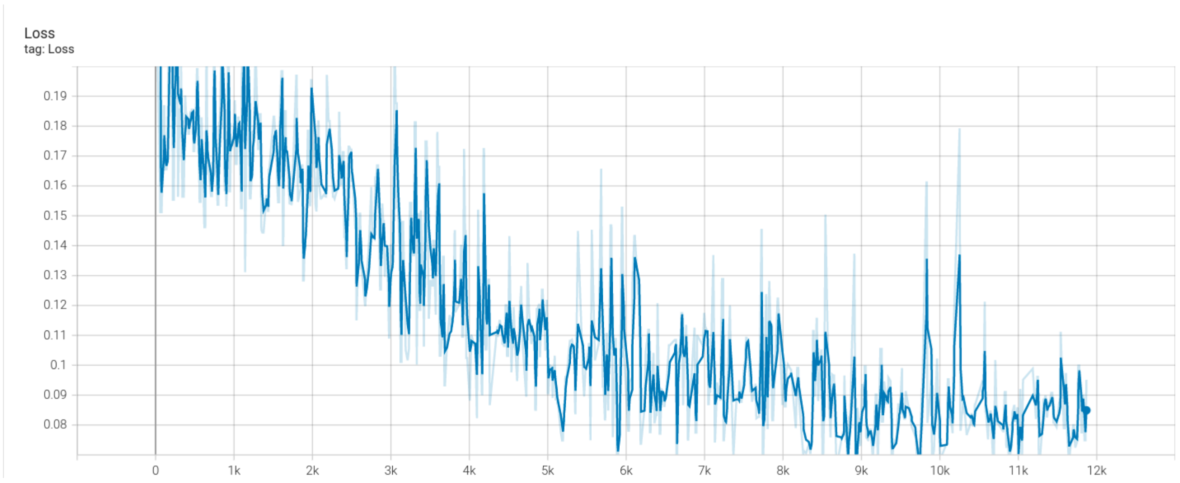

Figure 3: Loss Plot Validation Set

Precision, p = True Positive/(True Positive + False Positive)

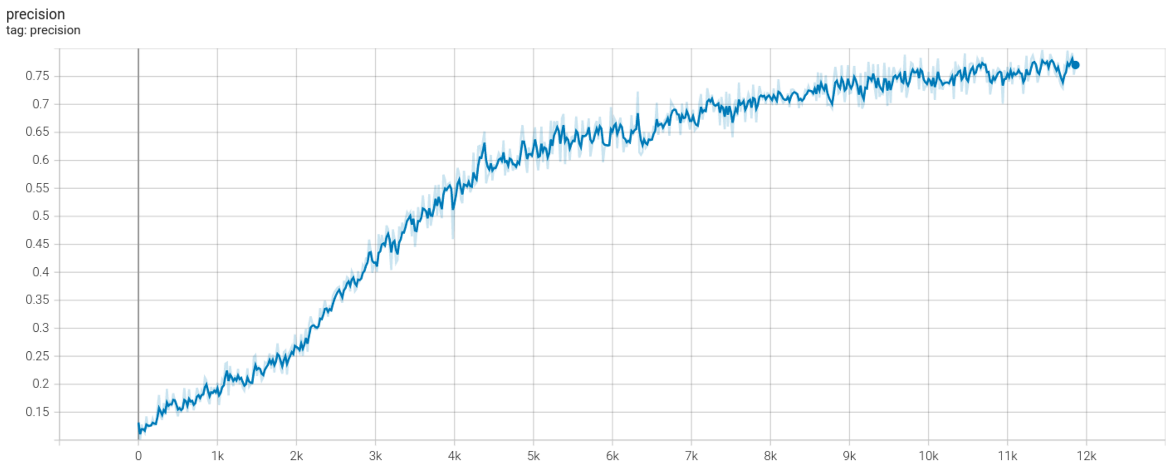

Figure 4: Precision

Recall, r = True Positive/(True Positive + False Negative)

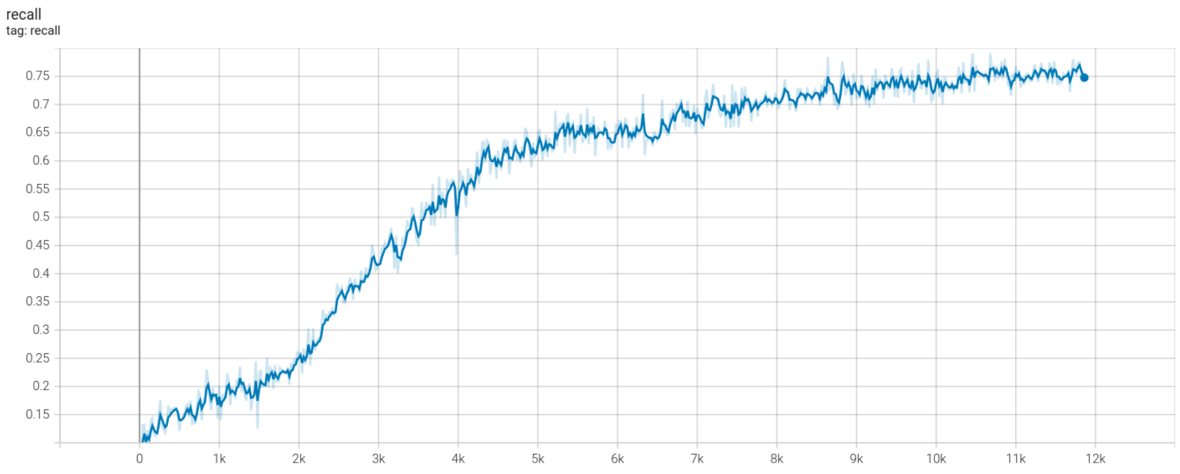

Figure 5: Recall

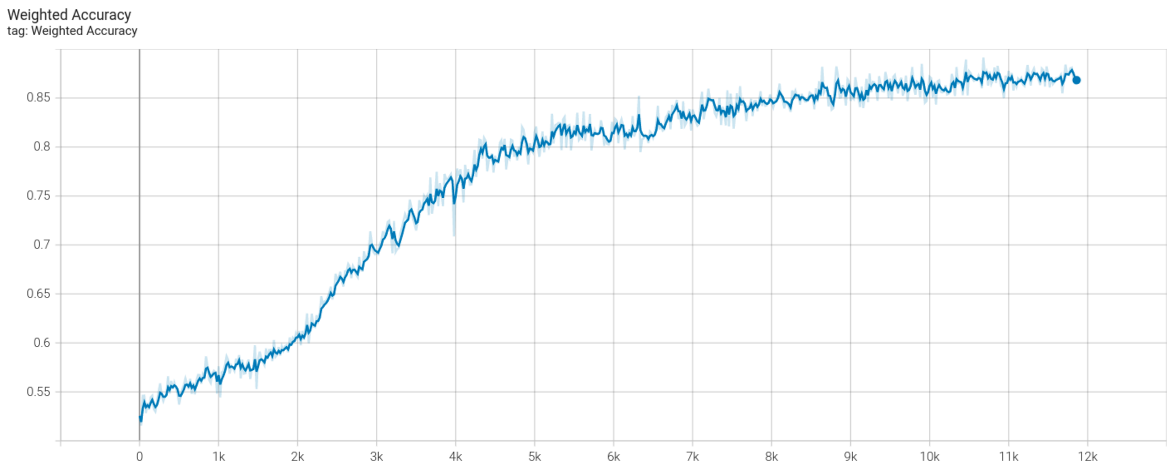

Figure 6: Weighted Accuracy

While running on the test set, the weighted accuracy we get is around 88

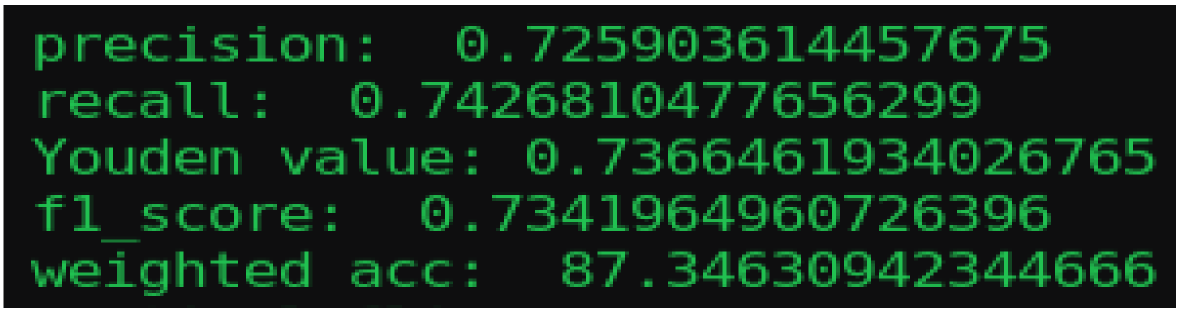
```
precision:  0.725903614457675
recall:  0.7426810477656299
Youden value: 0.7366461934026765
f1_score:  0.7341964960726396
weighted acc:  87.34630942344666
```

Figure 7: Measure Values on Test Set using the model trained on 11000 iterations

5. Limitations and Challenges

- Titan X GPU takes a long time approx. 20 hours to run on the datasets given in the original paper. We ran the model on ADA node having 2 GPUs and 10 CPUs. During training, the number of epochs run are lesser in our project as compared to the paper due to hardware limitations. The model took 2 hours to train on ADA for 1 epoch.
- The framework was complex to implement as one had to keep a check on the dimensionality at each layer of the neural network (I/p and O/p) and follow the essence of the paper while looking out for the limitations at hand.

6. Work Distribution

- Implementing DHN Model - Dhruv
- Implementing file for training DHN - Naimeesh
- Implementing file for evaluating DHN - Atharva
- Loss function and other helper functions - Soveet
- Code for Data preparation and Loading - Dhruv
- Mid-eval ppt making - All
- Final Presentation making - Soveet and Atharva
- Report making - Naimeesh, Dhruv and Soveet
- Obtaining GPU resources for running atleast 11k iterations - Atharva
- Bug fixing - Dhruv and Naimeesh