Adaptive RNN Tree for Large-Scale Human Action Recognition

Wenbo Li[1] Longyin Wen[2] Ming-Ching Chang[1] Ser Nam Lim[2] Siwei Lyu[1]
[1]University at Albany, SUNY [2]GE Global Research
{wli20,mchang2,slyu}@albany.edu {longyin.wen,limser}@ge.com

Abstract

In this work, we present the RNN Tree (RNN-T), an adaptive learning framework for skeleton based human action recognition. Our method categorizes action classes and uses multiple Recurrent Neural Networks (RNNs) in a tree-like hierarchy. The RNNs in RNN-T are co-trained with the action category hierarchy, which determines the structure of RNN-T. Actions in skeletal representations are recognized via a hierarchical inference process, during which individual RNNs differentiate finer-grained action classes with increasing confidence. Inference in RNN-T ends when any RNN in the tree recognizes the action with high confidence, or a leaf node is reached. RNN-T effectively addresses two main challenges of large-scale action recognition: (i) able to distinguish fine-grained action classes that are intractable using a single network, and (ii) adaptive to new action classes by augmenting an existing model. We demonstrate the effectiveness of RNN-T/ACH method and compare it with the state-of-the-art methods on a large-scale dataset and several existing benchmarks.

1. Introduction

Human action recognition is an important but challenging problem. With advances in low-cost sensors and real-time joint coordinate estimation algorithms [20], reliable 3D skeleton-based action recognition (SAR) is now feasible [1]. Recent methods [4, 13, 19, 22, 31] use the RNN models to advance the state-of-the-art performance of SAR.

Although much progress has been achieved, these methods are still facing two challenges. We term the first one as the *discriminative challenge*. In SAR, a human action is usually represented by the trajectories of approximately 20 key skeletal joints. This leads to a limited degree of freedom of 3D joint coordinates. As more action classes are packed into such a coordinate, the inter-class variations would be subtler. This causes the ambiguity among action classes, and makes decision boundaries between classes harder to determine. We term the second challenge as *adaptability*, *i.e.*, a desirable method should be able to handle new classes incrementally. Most previous methods handle new action classes by a time-consuming re-training of the whole model. Two methods [15, 18] use non-parametric models to handle new classes incrementally, but it is hard to adapt these methods to a large number of new classes.

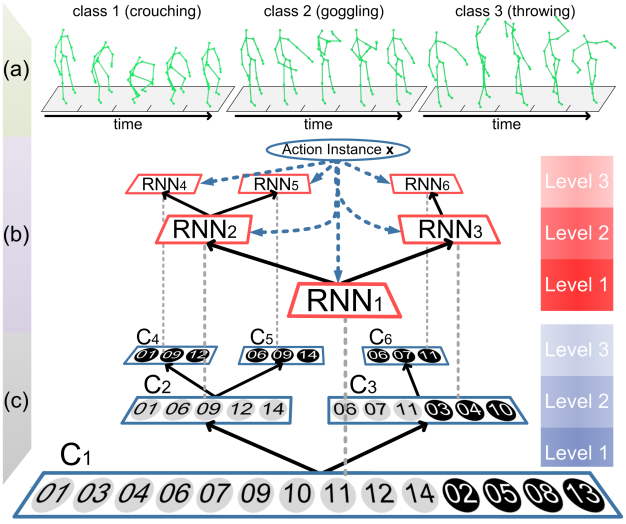

Figure 1: **Method overview.** *(a) Visualization of action instances from three action classes. (b) A three-level RNN Tree (RNN-T) associated with the learned Action Category Hierarchy (ACH) in (c). Each circle represents an action class. Grey circles represent* ambiguous *classes, and black circles represent* unambiguous *ones. Action classes in the same box form one action category.*

In this paper, we propose an adaptive learning framework that aggregates multiple discriminative RNNs hierarchically for large-scale SAR. We partition action classes into several *action categories*, and organize the action categories using a tree structure. We train a RNN model for each action category and co-train all individual RNN models, which are organized as a tree model (RNN-T) with the same structure as the action categories (see Figure 1). At run time, RNN-T recognizes actions via a hierarchical inference process, during which individual RNNs differentiate action classes with increasing confidence. Ambiguous decisions are deferred to sub-trees of RNNs where actions to be recognized can be effectively differentiated by finer-

grained RNN classifiers. The inference is finished when the action is recognized with a high confidence or a leaf node of RNN-T is reached. To handle increasing number of action classes, we further develop an incremental learning algorithm, so that new classes can be inserted into existing action categories in RNN-T, and the respective RNN sub-trees can be updated.

Further, we create a large-scale SAR dataset that has 140 action classes, which we term as `3D-SAR-140`, by aggregating and processing 10 existing smaller-scale datasets [3, 5, 6, 12, 14, 15, 16, 25, 27, 28]. Our dataset has significantly larger number of action classes than the existing SAR datasets: it is more than twice of the class number of the existing largest HDM05 dataset [14]. Experimental results on the `3D-SAR-140` dataset and the recently developed NTU dataset [19]show that RNN-T/ACH outperforms the current state-of-the-art SAR methods.

The main contribution of this work is four-fold. *(i)* We propose a novel, adaptive and hierarchical framework for fine-grained, large-scale SAR. Multiple RNNs are incorporated effectively in a tree-like hierarchy to mitigate the discriminative challenge using a divide-and-conquer strategy. *(ii)* We develop an effective learning procedure to build RNN-T to achieve high recognition accuracy and running efficiency. *(iii)* We design an incremental learning algorithm to make RNN-T adaptable to new classes and to significantly reduce the re-training time. *(iv)* We create a large-scale dataset, `3D-SAR-140`, with the largest number of action classes to-date, and produce a benchmark to evaluate existing SAR methods and RNN-T based method.

2. Related Works

SAR. Vemulapalli *et al*. [23, 24] modeled the 3D geometric interactions between body parts using transformations, which are represented as elements in a Lie group. Gowayyed *et al*. [7] represented 3D trajectories of body joints using the *histogram of oriented displacements* descriptor. Zanfir *et al*. [29] proposed a moving pose descriptor that considers both pose information and the differential quantities of body joints. Wang *et al*. [25] learned a subset of skeleton joints for each action class. Wu *et al*. [26] modeled transition dynamics of an action using the Hidden Markov Model and developed a hierarchal dynamic framework to learn high-level features to estimate emission probability. Zhao *et al*. [30] extracted structured streaming skeleton features to represent actions and constructed classifiers based on sparse coding features. Performance of these methods is bounded by their inability to model long-term evolution of discrete human poses, which are crucial for action recognition.

SAR using RNN. In recent years, RNN has become the most successful model for SAR. RNN [8] is a class of neural network whose neurons send feedback signals to each other. Along the time axis at each step, RNN accepts the current input together with the previous hidden state, and updates the current state via a set of non-linear activation functions. RNN is suitable for handling sequential data, because it can represent complex relations in the sequence and is robust to local distortions. One limitation of the original RNN is incapability in modeling long-term dependencies, due to the vanishing and exploding gradients [8]. This problem is alleviated with the *long short-term memory* (LSTM) cells [10] as a replacement of the traditional nonlinear units. A LSTM block contains a self-connected memory cell to store information across long duration. It also contains 3 recurrent gates (*i.e*., *input*, *forget*, and *output* gate) to adaptively adjust the "forgetting rate" of the old state.

For the SAR task, RNN typically performs multi-nominal classification and predicts an action class label at the end of the sequence. Most existing works are based on LSTM RNN . Existing RNN based methods fall into two groups. The first group uses RNN to model coordinates of body parts. Shahroudy *et al*. [19] proposed part-aware LSTM which divides the memory cell to sub-cells such that each sub-cell can learn long-term contextual representations of a body part, and the concatenation of sub-cells yields the final output. Zhu *et al*. [31] added a mixed-norm regularization term to the cost function of a LSTM network, which can learn the co-occurrence of discriminative joints. Du *et al*. [4] designed a hierarchical RNN for SAR, where the skeleton is divided into five parts and then fed into five subnets. Their representation is learned by fusing the outputs of higher layers. The second group focuses on developing gating schemes for LSTM to determine which part of the input is more important for recognition. Veeriah *et al*. [22] designed a "differential" gate emphasizing the information gain from salient motions. Liu *et al*. [13] proposed a "trust" gate to analyze input reliability, which provides insight for the modulation of memory cells in LSTM.

All these methods used a single RNN to model the whole action space. In contrast, in order to better resolve the discriminative challenge, our method exploits multiple RNNs in a hierarchy, with each RNN recognizing actions within one action category.

3. ACH and RNN-T for Action Recognition

We start with notations that will be used throughout the paper. Let $\{\mathbf{x}_i, y_i\}$ be the labeled action instance, where $\mathbf{x}_i$ is a sequence of 3D skeletal poses collected from a video sequence, and y_i is the label of $\mathbf{x}_i$ out of all N action classes. An *action category* C is defined as a set of action classes sharing similar characteristics. We use $\Re$ to represent the RNN model.

The goal of a SAR algorithm is to infer the class label of an action instance out of a large number of action classes. As stated in § 1, there are the ambiguity and adaptivity chal-

Algorithm 1 Learning of ACH and RNN-T

Input: C_1 including all N action classes
Output: ACH, RNN-T

Initialize ACH by embedding C_1, and marking C_1 as unvisited
Train $\Re_1$ for C_1
while $\exists\, C_i$ is unvisited and $|C_i| > \theta_l$ **do**
    • Generate candidate partitions for C_i (§ 3.1.1)
    • Pre-train RNNs for each candidate partition (§ 3.1.2)
    • Evaluate candidate partitions (§ 3.1.3)
    • Expand ACH and RNN-T based on the optimal partition
    • Fine tune the newly-added RNNs in RNN-T jointly (§ 3.1.4)
    Mark C_i as visited
end while

lenges to this problem. In particular, there exist ambiguity among action classes, and some actions are more difficult to distinguish the the others and require finer-grained classifiers. To this end, we first construct an *action category hierarchy* (ACH) to organize action categories based on the ambiguities of fine-grained action classes. Action categories at higher levels of ACH are more specific and difficult to recognize. Then, we build a RNN Tree (RNN-T) using the same structure of ACH (see Figure 1), with each individual RNN modeling a specific action category, and RNNs at higher levels of RNN-T are classifiers of the more fine-grained actions modeled by ACH.

This ambiguity-aware deferral strategy is implemented with divide-and-conquer as the following: *(i)* The root category C_1 of ACH initially contains all N action classes. The ambiguity of action classes in C_1 is estimated by recognizing their respective action instances in the training dataset. *(ii)* If the ambiguity between a specific class and others is low, its classification results are output directly, while the remaining classes with more inter-ambiguity are further clustered to form new action categories at the next level. This process repeats to produce a tree-like hierarchy of ACH, and an ensemble of RNNs can be trained at the same time. *(iii)* During the run time, ambiguous decisions are deferred to sub-trees of RNNs where the action instance to be recognized can be effectively differentiated by higher level individual RNN classifiers in the RNN-T model. In the following, we define the children of the i-th action category C_i in ACH as C_i^j with $C_i^j \subseteq C_i$. We use $\Re_i$ to denote the RNN that corresponds to C_i. Similarly, the children of $\Re_i$ is denoted as $\Re_i^j$.

RNN-T/ACH is similar to a decision tree (DT) with RNNs as the base classifiers but with two important distinctions. *(i)* The base classifiers in a DT are learned *separately*, but the RNN classifiers in RNN-T/ACH are co-trained following the tree structure. *(ii)* Action classes contained in different action categories of ACH can overlap, *e.g.*, both C_2 and C_3 in Figure 1(c) contain action class `06`. Therefore, unlike a DT where a classification error is irrecoverable once a branch is reached, RNN-T/ACH allows multiple sub-trees to output the same class label. In the following, we describe in detail the learning of RNN-T/ACH (§ 3.1), and how this model can be applied to SAR (§ 3.2).

3.1. Learning of ACH and RNN-T

As ACH and RNN-T have the same tree structure, they are learned jointly from the labeled training data with a level-by-level scheme. The learning algorithm for RNN-T and ACH is as summarized in Algorithm 1. Starting with the root action category C_1, each action category is successively divided into finer categories at the next level, where a RNN is trained for each newly created action category.

The **partition** of an action category is performed in four steps, which are summarized here and described in details in the subsequent sessions. First we identify all *ambiguous classes* in an action category, which are the classes whose labels cannot be confidently determined with the RNN model of the current level. These classes are considered difficult to distinguish and are further divided into sub-categories to form new action categories of the next level. Instead of using a fixed partition, we generate multiple candidate partition hypotheses of the ambiguous classes by repeatedly running a *spectral clustering* algorithm [2] (§ 3.1.1). For each candidate partition, a set of RNNs are pre-trained independently (§ 3.1.2). The optimal partition is then determined based on a performance evaluation metric, which is used to generate new action categories at the next-level (§ 3.1.3). RNNs corresponding to the newly generated action categories are fine tuned jointly (§ 3.1.4). After one level of ACH is created, the same process is repeated for the next level if necessary. The process completes when all action classes are classified by the RNNs in RNN-T with high confidence, or the number of action classes in all leaf action categories is below a preset threshold.

3.1.1 Generation of Candidate Partitions

For each action category, if it contains any ambiguous action class, it needs to be further divided into children action categories at the next level of the ACH. Specifically, if we are to process action category C_i, we first identify ambiguous classes within C_i using its corresponding RNN model $\Re_i$. For each action class $c_s \in C_i$, we compute the F-scores of the training and validation datasets, respectively, based on the recognition results generated by $\Re_i$. If these two F-scores are greater than a pre-determined threshold θ_c, then c_s is marked as an ambiguous class.

After all action classes in C_i are processed, all ambiguous classes are put together as $\widehat{C}_i$. If $|\widehat{C}_i| \leq \theta_l$, C_i is not further partitioned, where θ_l is the target size of a leaf action category in the ACH. Otherwise, we generate at most

$n = \lfloor N/\theta_l \rfloor$ different partitions of $\widehat{C}_i$ in two steps. First, we calculate the *confusion matrix* based on recognition results of RNN $\Re_i$ on the validation dataset. Then, we use spectral clustering [2] to generate partitions of the action category using the confusion matrix as their affinities. We run the clustering algorithm n times, each time split the action category into k disjoint clusters, each cluster is denoted as $\tilde{C}_i^{k,j} \subseteq \widehat{C}_i$ for $1 \leq k \leq n$ and $1 \leq j \leq k$. Note that we have $\tilde{C}_i^{1,1} = \widehat{C}_i$.

This disjoint partition scheme does not allow error recovery when misclassification occurs during the partition of the ACH. To improve the fault tolerance of RNN-T/ACH, we allow ambiguous action classes to be associated with more than clusters. Specifically, for each cluster $\tilde{C}_i^{k,j}$, we compute a *misclassification likelihood* $p_i^j(s)$ for each class $c_s \notin \tilde{C}_i^{k,j}$. We use X_s to denote the set of action instances in c_s. $\mathbf{1}(\subset)$ is an indicator function that outputs 1 if $\subset$ is true, and 0 otherwise, then $p_i^j(s)$ is defined as:

$$p_i^j(s) = \frac{\sum_{\mathbf{x} \in X_s, c_s \notin \tilde{C}_i^{k,j}} \mathbf{1}(y' \in \tilde{C}_i^{k,j})}{|X_s|}. \quad (1)$$

In other words, $p_i^j(s)$ is the fraction of action instances in c_s that are misclassified into $\tilde{C}_i^{k,j}$. If $p_i^j(s) > \theta_o$, where θ_o is a pre-determined threshold, the action instance of c_s is likely to be misclassified by RNN $\Re_i$ into $\tilde{C}_i^{k,j}$, then it is added to the child action category of $C_i^{k,j}$, as $C_i^{k,j} = \tilde{C}_i^{k,j} \cup \{c_s\}$.

3.1.2 Pre-training RNNs Using Candidate Partitions

To maximize the recognition performance of RNN-T, it will be ideal if all individual RNNs in the RNN-T model can be trained jointly. However, the time complexity of training RNNs jointly grows with the number of RNNs. Moreover, the candidate partitions described in §3.1.1 must be followed by the training of RNNs, such that each candidate partition can be evaluated using respective RNNs to find the optimal partition. Thus, for n candidate partitions, total RNN training time increases quadratically, *i.e.*, $O(\frac{n(n+1)}{2})$. To reduce the training complexity, we use a trade-off method that starts with independently pre-train the individual RNNs and then fine-tunes them jointly.

For each candidate partition $C_i^{k,j}$, we train a set of RNNs $\Re_i^{k,j}$, *i.e.*, to obtain its parameters W using training data. We initialize W using weights of its parent model $\Re_i$, except those on the output layer (which takes random values). We use $\mathbf{x}_r$ to represent the r-th action instance in the training set and y_r to represent the ground truth label of $\mathbf{x}_r$. $\Re_i^{k,j}$ is trained by minimizing the log likelihood loss function $-\sum_r \ln p(y_r|\mathbf{x}_r)$, where $p(y_r|\mathbf{x}_r)$ is the output of the *softmax* function of $\Re_i^j$ which represents the probability of $\mathbf{x}_r$ being labeled as y_r, using the *stochastic gradient descent* (SGD) method with gradient computed using the back-propagation through time (BPTT) algorithm [8].

3.1.3 Evaluate Candidate Partitions

We use the individually trained RNNs to choose an optimal partition for category C_i. To this end, we first build a two-level temporary ACH CH_k, with C_i as the root at the first level and the k-th candidate partition $\{C_i^{k,j}\}_{j=1}^k$ at the second level as the children of C_i. RNN $\Re_i$ corresponding to C_i, and a set of RNNs $\{\Re_i^{k,j}\}_{j=1}^k$ corresponding to the j-th candidate partition are organized in a two level RNN-tree (RNN-ST_k) with the same structure as CH_k. Note that within CH_k, an ambiguous class c_s ($\in C_i$) may belong to multiple sub-categories $\{C_i^{k,j}\}$. Thus we maintain a lookup table to effectively defer c_s to the desired subcategory. Specifically, for candidate partition $\{C_i^{k,j}\}_{j=1}^k$, the lookup table $f_{i,k}(\cdot)$ of C_i is built upon $\{C_i^{k,j}\}_{j=1}^k$ and its corresponding disjoint partition $\{\tilde{C}_i^j\}_{j=1}^k$ (see § 3.1.1), where $\tilde{C}_i^j \subseteq C_i^{k,j}$. As a result, for a predicted label y', if $y' \in \tilde{C}_i^j$, we have $f_{i,k}(y') = C_i^{k,j}$.

Next, we introduce a metric $\mathcal{R}$ to evaluate the reliability of each candidate partition inspired by the splitting of nodes in a decision tree [17], which is defined as:

$$\mathcal{S} = \underbrace{\mathcal{A}^t + \mathcal{A}^v + \min\left(\frac{\mathcal{A}^t}{\mathcal{A}^v}, \frac{\mathcal{A}^v}{\mathcal{A}^t}\right)}_{accuracy} - \lambda \underbrace{\exp\left(\frac{H}{N} \log N_l\right)}_{inefficiency}, \quad (2)$$

where N_l is the number of leaf nodes, H is tree depth, N is the number of all classes, and λ balance accuracy and inefficiency terms. The accuracy term consists of three parts, *i.e.*, $\mathcal{A}^t$ and $\mathcal{A}^v$ are the training and validation classification accuracy of each candidate partition, computed by feeding the training and validation data back to RNN-ST_k, and $\min(\frac{\mathcal{A}^t}{\mathcal{A}^v}, \frac{\mathcal{A}^v}{\mathcal{A}^t})$ measures the stability between $\mathcal{A}^t$ and $\mathcal{A}^v$, which ensures that RNN-T will not yield recognition accuracy with large variations when applied to training and validation datasets. The *inefficiency* term penalizes trees that are deep but with only a few leaves. Note that the usage of the inefficiency term potentially reduces the risk to select an over-fitted tree structure of RNN-T/ACH.

Thus, for the current category C_i, we calculate the score $\mathcal{S}_k$ for each candidate partition. Note that $\mathcal{S}_1$ corresponds to the case when no partition of C_i is performed. After that, we determine the optimal partition by maximizing the reliability values from both *(i)* all candidate partitions and *(ii)* no partition cases, *i.e.*, $m = \text{argmax}_k \mathcal{S}_k$. If $m = 1$, we do not divide the current category C_i. Otherwise, we divide C_i into finer categories based on the m-th partition strategy at the next level. Correspondingly, RNNs $\{\Re_i^{k,j}\}_{j=1}^m$ are organized into RNN-T and the cross level lookup table $f_i(\cdot)$ is updated accordingly.

3.1.4 Joint Fine-tuning of RNNs

We fine tune the new RNNs $\{\Re_i^j\}_{j=1}^m$ and their parent $\Re_i$ jointly to achieve higher classification accuracy. Specifi-

cally, we add a new term in the RNN objective function to reduce the risk of deferring ambiguous label prediction to a wrong action category, *i.e.*,

$$L_\Phi(W) = -\sum_{r=1}^{|\Phi|} \ln\{\mathbf{1}(\text{ch}_i(y_r{}') = \emptyset)p(y_r|\mathbf{x}_r)+$$
$$\sum_{j=1}^{m}[\mathbf{1}(\text{ch}_i(y_r{}') = C_i^j)\sum_{s=1}^{|C_i^j|}\mathbf{1}(c_s = y_r)p_j(c_s|\mathbf{x}_r)]\}+$$
$$\frac{\alpha}{|\Phi|}\sum_{r=1}^{|\Phi|}\mathbf{1}(\text{ch}_i(y_r{}') \neq \text{ch}_i(y_r) \wedge \text{ch}_i(y_r) \neq \emptyset \wedge \text{ch}_i(y_r{}') \neq \emptyset),$$

where W is the learnable weights of $\{\Re_i^j\}_{j=1}^m$ and their parent $\Re_i$, Φ is the training set corresponding to C_i, y_r is the ground truth label of $\mathbf{x}_r$, and $y_r{}'$ is the label of $\mathbf{x}_r$ that is predicted by $\Re_i$. The children of C_i are denoted as $\{C_i^j\}_{j=1}^m$ that corresponds to $\{\Re_i^j\}_{j=1}^m$. If $y_r{}'$ refers to an ambiguous class, $y_r{}'$ will be deferred via $\text{ch}_i(\cdot)$ to a specific child of C_i; otherwise, there will be no deferral and $\text{ch}_i(y_r{}')$ will be set to $\emptyset$. $p(y_r|\mathbf{x}_r)$ and $p_j(c_s|\mathbf{x}_r)$ are the outputs of the softmax function in $\Re_i$ and $\Re_i^j$, respectively. Parameter α balances the terms in the objective function and is set to 10 in our current implementation. Similar to the RNN pre-training (§ 3.1.2), $L_\Phi(W)$ is optimized by SGD, where the gradients are computed by BPTT.

3.2. Recognition using RNN-T/ACH

Applying RNN-T/ACH to SAR leads to an iterative algorithm that traverses the RNN-T model. For an input skeleton sequence $\mathbf{x}$, the recognition starts at the root level, where the root RNN $\Re_1$ (corresponding to C_1) generates its classification result $y_1{}'$. If $y_1{}'$ refers to an unambiguous class, $y_1{}'$ is output directly and the recognition process is completed. Otherwise, $y_1{}'$ is deferred via the lookup table $f_1(\cdot)$ to a specific child C^j of C_1 for finer classification using $\Re^j$. This process continues until $\mathbf{x}$ is recognized with high confidence (*i.e.*, the predicted label of $\mathbf{x}$ refers to an unambiguous class), or a leaf node of RNN-T produces the final classification result.

4. Incremental Learning

When RNN-T/ACH encounters action classes that are not presented in training, we augment it to include the new classes using an *incremental learning algorithm* to avoid a time-consuming re-training of the entire model. The key is to transfer information from the existing RNN-T/ACH model to handle the limited training data of the new classes. Specifically, the topology of the existing ACH, which is represented by the inter-category relations encoded by the ambiguous class deferral lookup tables (§ 3.1.3), is preserved in the augmented model, and the network structure shared by individual RNNs in RNN-T (§ 3.1.2) is also inherited by the new RNN-T model. Our incremental learning algorithm updates ACH and RNN-T level-by-level from the root level using two main procedures: (a) insert new classes into the action categories with similar actions; (b) update ACH and RNN-T to reflect the change in structure. Figure 2 illustrates a update example.

We insert new classes into action categories where similar classes exist in ACH. All new classes are initially inserted into the root action category C_1. We then traverse the tree structure of ACH to find appropriate action categories for the new classes. Specifically, when the process reaches an action category C_i (with children $\{C_i^j\}$), we estimate the likelihood $\tilde{p}_i^j(s)$ that an action instance in new class c_s is classified by RNN $\Re_i$ into each child of C_i:

$$\tilde{p}_i^j(s) = \frac{\sum_{\mathbf{x}\in X_s, c_s\in C_i}\mathbf{1}(y' \in C_i^j)}{|X_s|}, \tag{3}$$

where the numerator counts how many action instances of c_s are classified into C_i^j. X_s represents the action instance set of c_s. If $\tilde{p}_i^j(s) > \theta_o$, c_s is inserted into C_i^j, where θ_o is a threshold defined in § 3.1.1. As such, possibly multiple children of C_i will process c_s in parallel subsequently. The process continues until a leaf node is reached.

After new action classes are inserted into ACH, action categories $\{C_i\}$, lookup tables $\{f_i(\cdot)\}$ and the RNN modules $\{\Re_i\}$ in RNN-T are then updated in a similar level-by-level fashion. We traverse the tree structure of ACH and RNN-T. When the process reaches an action category C_i, we update $\Re_i$ to recognize new classes in C_i. Then, we use the updated $\Re_i$ to identify the ambiguity of these new classes as in § 3.1.1. If there exists any new action class in C_i being identified as unambiguous, we remove such unambiguous classes from the offsprings of C_i in ACH. Finally, we update the lookup table $f_i(\cdot)$, by incrementally updating the lookup table for minor changes, but reconstruct it when significant changes occur in the overall structure. We measure the degree of change occurring in C_i using the increment ratio of ambiguous classes in it, which is defined as $\tau = \frac{n_{new}}{n_{old}}$ [1]. n_{new} and n_{old} represent the number of new ambiguous classes and old ambiguous classes, respectively. We define $\theta_r = h \cdot \exp(1-h)$ as a threshold for τ, with h indicating the depth of the level that C_i resides. $\tau < \theta_r$ means that the change in C_i is minor, so we just update $f_i(\cdot)$ by enabling the deferral for each new ambiguous class c_s to a specific child C_i^j with the highest likelihood defined in (3). When $\tau \geq \theta_r$, it means significant changes have occurred to the composition of C_i due to the new classes, thus, we rebuild the entire sub-tree structure of RNN-T/ACH starting from C_i as described in § 3.1, and update the lookup table accordingly.

[1] If $n_{new} > 0$ and $n_{old} = 0$, it means a drastic change. If $n_{new} = 0$ and $n_{old} = 0$, we set τ as 0.

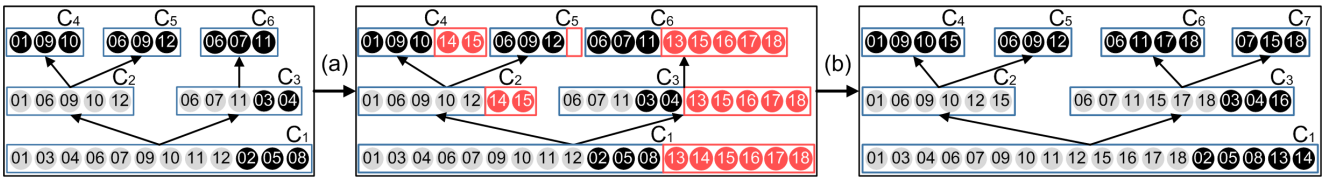

Figure 2: *An example of ACH after each incremental learning procedure. Red circles represent new action classes. (a)* **Insert new classes:** *All action categories accommodate new classes except C_5, which does not contain similar classes to the new ones. (b)* **Update ACH and RNN-T:** *Minor changes occur in C_1, C_2, and C_4, and their corresponding RNNs are incrementally updated. The sub-tree starting from C_3 is rebuilt due to drastic changes.*

5. Experiments

We evaluate RNN-T/ACH model for SAR problem with two test settings: *(i)* using a fixed number of action classes (§ 5.2), and *(ii)* using increasing number of classes over time (§ 5.3). For scenario *(i)*, we only consider the classification accuracy for evaluation. Test scenario *(ii)* is used to evaluate our incremental learning algorithm, so we consider both accuracy and the re-training time for evaluation. All results are reported based on the implementation using a single CPU core (3.4GHz) on an Intel Xeon E5-2687W v2 machine with 128GB RAM. The four parameters λ, θ_c, θ_o, θ_l described in § 3.1.1 are chosen as follows. Since inefficiency grows exponentially with ACH levels, we set the balancing parameter λ in (2) to be a small value $\lambda = 0.03$. The threshold θ_c is empirically set to be 0.85 to determine whether a class is ambiguous.

We construct 6 variants of RNN-T using different individual RNN modules, namely, *uni-directional vanilla RNN* (URNN), *bi-directional vanilla RNN* (BRNN), *uni-directional RNN with LSTM* (URNN-L), *bi-directional RNN with LSTM* (BRNN-L), *hierarchically bidirectional RNN* (HBRNN-L) [4], *uni-directional RNN with 2 layers of LSTM* (URNN-2L). The code of HBRNN-L [4] is available, while the other RNNs are re-implemented using RNNLIB [9].

Concerning the input to the RNNs, similar to [4], skeletal joints are divided into five parts (*i.e.*, four limbs and one trunk) as the input to HBRNN-L. For the rest of RNN models, we follow [22] to extract four features (positions, angles, offsets, pairwise joint distances) from the skeletal joints, and concatenate them to create a 310 dimensional feature vector per frame. The network architecture and other configurations of RNN are set according to [4] and [22]. See supplemental materials for more details. Among these RNN modules, URNN-2L is the largest model, producing the best recognition accuracy, however, it requires the longest training time. We made a trade-off between the recognition accuracy and training efficiency, and chose HBRNN-L as the major RNN module in RNN-T/ACH for the extensive parameter selection experiments and the validation experiments for incremental learning with new classes.

5.1. Datasets

We create a new dataset with 140 diverse action classes by aggregating all distinct classes from 10 existing datasets, which we name `3D-SAR-140`. The 10 existing datasets are CMU Mocap [3] (23), ChaLearn Italian [5] (20), MSRC-12 Gesture [6] (12), MSR Action3D [12] (20), HDM05 [14] (65), Kintense [15] (10), Berkeley MHAD [16] (12), MSR Daily Activity 3D [25] (13), UTKinect-Action [27] (10), and ORGBD [28] (7), where the number of classes are shown in the parentheses. The class list is presented in the supplemental materials.

We re-organize and standardize all attributes across 10 datasets to form `3D-SAR-140`, such that the number of sequences per class is 28 on average, and the frame rate is normalized to 20 frames-per-second (FPS), and the human skeleton is represented by 20 skeletal joints (see supplemental material for details). We partition 60% of the `3D-SAR-140` as the training set, 20% as the validation set, and the remaining 20% as the testing set. `3D-SAR-140` is a challenging benchmark due to two factors: *(i)* a large variety of movements and dynamics in various contexts are included, where fine-grained recognition is required; *(ii)* video length for individual actions varies significantly (ranging from 5 to 800 frames) within or across classes. `3D-SAR-140` dataset. The dataset is available for download from `http://www.cs.albany.edu/cvml/downloads.html`.

We also evaluate our method on NTU RGB+D Dataset [19], a new dataset which contains both single-person actions and mutual actions. This dataset is collected by Kincet v2 and contains more than 56 thousand sequences and 4 million frames. A total of 60 different action classes including 40 daily actions (*e.g.*, drinking, eating, reading, *etc.*), 11 mutual actions (*e.g.*, punching, kicking, hugging, *etc.*), and 9 health-related actions (*e.g.*, sneezing, staggering, falling down, *etc.*) are performed by 40 subjects aged between 10 and 35. The 3D coordinates of 25 joints are provided in this dataset. The large intra-class and view point variations make this dataset challenging.

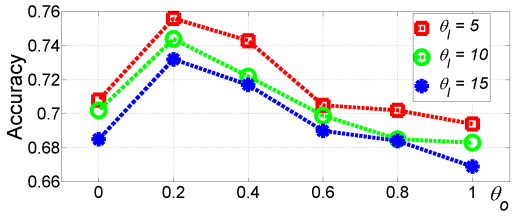

Figure 3: *Classification accuracy against θ_o and θ_l.*

5.2. Fixed Action Classes

To evaluate RNN-T/ACH in multiple aspects, we switch key features (*i.e.*, EJR, IP, and FT) on and off to demonstrate the effectiveness of the individual components of RNN-T/ACH. We further compare RNN-T/ACH with DT with RNN base classifiers and several variants of RNN-T/ACH with 5 baselines (URNN, URNN-L, BRNN, BRNN-L, URNN-2L) and 8 state-of-the-art methods (Table 1) in solving SAR.. Whenever possible, we use codes provided by the original authors of the corresponding work, with parameters chosen using a grid search around default parameters. All methods based on RNN-T/ACH are denoted with suffix "-T". We select the parameters of all methods on the validation set of the `3D-SAR-140` dataset.

Comparison with baselines and state-of-arts. As shown in Table 1, our method achieves significant performance gain over 5 baselines and 8 state-of-the-art methods. This maybe due to the ambiguity-aware deferral and divide-and-conquer approach used in RNN-T/ACH. In particular, URNN-2L-T yields the best performance, and the accuracy is improved by 13.6% compared to HBRNN-L-T. The main reason is that the base classifier URNN-2L is much larger and complicated than HBRNN-L, with 14 times more parameters than HBRNN-L.

Action category division vs. RNN decision tree. Parameters θ_l and θ_o control the category division in § 3.1. We vary θ_l among $\{5, 10, 15\}$ and θ_o among $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$, while keeping other parameters fixed, to investigate their impacts on the accuracy. Setting $\theta_o = 1$ disables the overlapping between action categories, which makes RNN-T/ACH degenerate to a RNN based decision tree. The best results are obtained with $\theta_l = 5$ and $\theta_o = 0.2$ as shown in Figure 3, which are used in the subsequent experiments.

Early jump-out of recognized action classes (EJR). To study the effect of EJR in ordinary RNN-T, we build a standalone HBRNN-L-T with $\theta_c = \infty$ which essentially implements EJR by treating all classes to be ambiguous — and hence all classes need to be deferred to subtrees as discussed in § 3.1.1. The resulting accuracy decreases from 0.756 to 0.700, which shows that the early output of confidently recognized classes is advantageous for efficiency and recognition performance.

Inefficiency penalization (IP). To verify the inefficiency penalization term in (2) designed to prevent over-fitting, we build a standalone HBRNN-L-T without inefficiency penalization. Consequently, a 6-level, 26-category ACH is generated, which is more complex than the 4-level, 22-category one generated by using the inefficiency penalization. As shown in Table 1, if we remove IP, accuracy drops from 0.756 to 0.697 but running time is 1.32 times faster, which may be due to over-fitting caused by the complex structure.

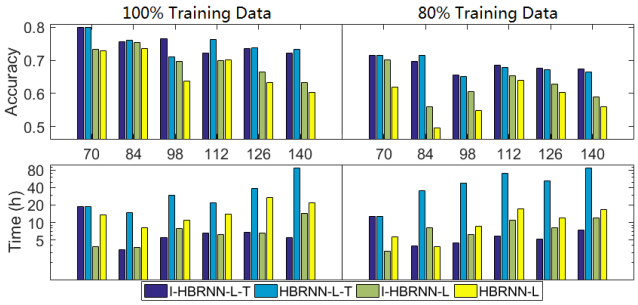

Figure 5: *Incremental learning comparison results. "I-" stands for* incremental.

Methods	Accur.
URNN	0.296
URNN-L	0.665
BRNN	0.643
BRNN-L	0.672
URNN-2L	0.866
RR [24]	0.723
HBRNN-L [4]	0.604
CHARM [11]	0.618
DBN-HMM [26]	0.601
Lie-group [23]	0.745
HOD [7]	0.657
MP [29]	0.203
SSS [30]	0.253

Our Methods	Accur.
URNN-T	0.539
URNN-L-T	0.743
BRNN-T	0.705
BRNN-L-T	0.751
URNN-2L-T	**0.892**
HBRNN-L-T (4 levels)	0.756
HBRNN-L-T (3 levels)	0.750
HBRNN-L-T (2 levels)	0.735
HBRNN-L-T (1 level)	0.604
HBRNN-L-T w/o EJR	0.700
HBRNN-L-T w/o IP	0.697
HBRNN-L-T w/o FT	0.733

Table 1: *Recognition results on* `3D-SAR-140` *dataset. See text that "-L" stands for* LSTM*, "-T" stands for* RNN-T.

Fine-tuning (FT). The joint fine-tuning in § 3.1.4 is another factor that RNN-T/ACH is superior than the RNN-based decision tree (where RNNs are trained separately). As shown in Table 1, such fine-tuning increases performance from 0.733 to 0.756, demonstrating that fine-tuning in RNN co-training is important to improve the performance.

Increasing levels of RNN-T. Table 1 shows that as the level of RNN-T/ACH increases, accuracy increases monotonically until saturation close to 4 levels. This demonstrates that a single RNN is not sufficient for large-scale action recognition, and for the module HBRNN-L, a 4-level RNN tree is a good trade-off between performance and efficiency.

Comparison on existing datasets. To provide broader contexts of RNN-T/ACH, we compare it with the state-of-the-art methods on the 10 existing datasets that are used to create `3D-SAR-140` dataset. We follow existing methods to setup experiments (see the supplementary for more details). Figure 4 summarizes the results. As the number of classes decreases, our URNN-2L-T outperforms the compared methods, though the advantage is less obvious than the large-scale case. The performance of RNN-T/ACH is

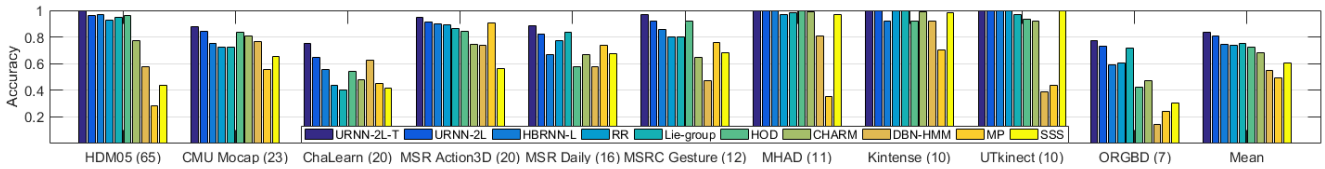

Figure 4: *Recognition results on existing datasets. The number of action classes of each dataset is shown behind its name. On average, our URNN-2L-T outperforms URNN-2L by* 3.2%*, and outperforms RR [24] by* 8.64%*.*

Methods	CS Accur.	CV Accur.
HBRNN-L [4]	0.591	0.640
Part-aware LSTM [19]	0.629	0.703
Deep RNN [19]	0.563	0.641
Deep LSTM [19]	0.607	0.673
ST-LSTM [13]	0.692	0.777
STA-LSTM [21]	0.734	0.812
URNN-2L	0.730	0.809
URNN-2L-T	**0.746**	**0.832**

Table 2: *Comparison on the NTU dataset with Cross-Subject (CS) and Cross-View (CV) settings.*

restricted by insufficient training data in the cases of MSR Daily and ORGBD. Furthermore, as shown in Table 2, we compare RNN-T/ACH with the most recent state-of-the-art LSTM based methods on the recently proposed NTU dataset (including 60 action classes), following a common protocol in [19]. The results of others' methods in Table 2 are directly taken from their original papers. The results show that RNN-T/ACH performs slightly better than the others based on URNN-2L without the sophisticated network structure design.

5.3. Including New Action Classes

To create an incremental learning process, we start from the data with 70 classes from `3D-SAR-140` dataset and incrementally add 14 random new classes at a step, until 140 classes are reached. To create a realistic simulation, we assume that the first 70 classes were trained from scratch using 100% of the training data. Then, we vary the percentage of the training data used for new classes as 100% and 80%, to see the impact of training data on the re-training time and accuracy. We compare two incremental learning methods I-HBRNN-L-T and I-HBRNN-L, and two from-scratch learning methods HBRNN-L-T and HBRNN-L, where I-HBRNN-L is initialized with weights from HBRNN-L.

Figure 4 shows that I-HBRNN-L-T achieves higher accuracy and less re-training time than I-HBRNN-L and HBRNN-L. I-HBRNN-L-T achieves similar accuracy compared to HBRNN-L-T, but with significantly less re-training time. We highlight two observations as the number of class increases: *(i)* both I-HBRNN-L-T and HBRNN-L-T yield stable accuracy, while the performances of I-HBRNN-L and HBRNN-L fluctuate, which demonstrates the advantage of RNN-T; *(ii)* the re-training time of I-HBRNN-L-T is relatively short, which demonstrates the effectiveness of our incremental learning algorithm. Furthermore, the fewer training data, I-HBRNN-L-T still outperforms three baselines in both performance and running efficiency.

6. Conclusion

We describe a new method for skeleton-based action recognition (SAR) using the RNN tree (RNN-T) model and its associated action category hierarchy (ACH). We show that organizing multiple RNNs into a tree structure together with the learned ACH leads to an effective and adaptive framework for large-scale and fine-grained SAR task. The RNN-T/ACH method addresses two main challenges in large-scale action recognition: *(i)* ability to distinguish fine-grained action classes that are intractable using a single network, and *(ii)* adaptability to new action classes by augmenting an existing model. We demonstrate the noticeable performance improvement against state-of-the-art methods on `3D-SAR-140` and several public benchmarks.

There are a few research directions that we would like to further improve the current work. First, our current method uses fixed structure for each action category, but it will be beneficial to differentiate the RNN structures adaptively in each action category, to capture richer space-time characteristics. As such, we will explore methods to optimize the structure of individual RNNs for each. Second, we plan to extend our dataset to include more diverse and challenging action classes for comprehensive evaluation. Last, the RNN-T/ACH model suggests that we can combine the recurrent neural network model with the recursive structure defined by the tree, thus a recursive recurrent neural network model may be more suitable to SAR and other related tasks. We plan to further pursue this direction in the future.

Acknowledgement. This material is based upon work partially supported (Siwei Lyu and Wenbo Li) by the National Science Foundation under *National Robotics Initiative* Grant No. IIS-1537257. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank the three anonymous reviewers for their constructive comments.

References

[1] J. K. Aggarwal and L. Xia. Human activity recognition from 3D data: A review. *PR Letters*, 48:70–80, 2014.

[2] W. Chen, Y. Song, H. Bai, C. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *TPAMI*, 33(3):568–586, 2011.

[3] CMU. CMU graphics lab motion capture database. http://mocap.cs.cmu.edu/, 2013.

[4] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *CVPR*, pages 1110–1118, 2015.

[5] S. Escalera, J. Gonzàlez, X. Baró, M. Reyes, O. Lopes, I. Guyon, V. Athitsos, and H. J. Escalante. Multi-modal gesture recognition challenge 2013: Dataset and results. In *ICMI*, pages 445–452, 2013.

[6] S. Fothergill, H. M. Mentis, P. Kohli, and S. Nowozin. Instructing people for training gestural interactive systems. In *CHI*, pages 1737–1746, 2012.

[7] M. A. Gowayyed, M. Torki, M. E. Hussein, and M. El-Saban. Histogram of oriented displacements (HOD): describing trajectories of human joints for action recognition. In *IJCAI*, pages 1351–1357, 2013.

[8] A. Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385 of *Studies in Computational Intelligence*. 2012.

[9] A. Graves. RNNLIB: A recurrent neural network library for sequence learning problems. http://sourceforge.net/projects/rnnl/, 2013.

[10] S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. In *ACCV*, pages 473–479, 1996.

[11] W. Li, L. Wen, M. C. Chuah, and S. Lyu. Category-blind human action recognition: A practical recognition system. In *ICCV*, pages 4444–4452, 2015.

[12] W. Li, Z. Zhang, and Z. Liu. Action recognition based on a bag of 3D points. In *CVPRW*, pages 9–14, 2010.

[13] J. Liu, A. Shahroudy, D. Xu, and G. Wang. Spatio-temporal lstm with trust gates for 3D human action recognition. In *ECCV*, pages 1–8, 2016.

[14] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database HDM05. Technical Report CG-2007-2, Universität Bonn, 2007.

[15] S. M. S. Nirjon, C. Greenwood, C. Torres, S. Zhou, J. A. Stankovic, H. Yoon, H. Ra, C. Basaran, T. Park, and S. H. Son. Kintense: A robust, accurate, real-time and evolving system for detecting aggressive actions from streaming 3D skeleton data. In *PerCom*, pages 2–10, 2014.

[16] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Berkeley MHAD: a comprehensive multimodal human action database. In *WACV*, pages 53–60, 2013.

[17] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[18] K. K. Reddy, J. Liu, and M. Shah. Incremental action recognition using feature-tree. In *ICCV*, pages 1010–1017, 2009.

[19] A. Shahroudy, J. Liu, T. Ng, and G. Wang. NTU RGB+D: a large scale dataset for 3D human activity analysis. In *CVPR*, pages 1010–1019, 2016.

[20] J. Shotton, T. Sharp, A. Kipman, A. W. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *CACM*, 56(1):116–124, 2013.

[21] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *AAAI*, pages 4263–4270, 2017.

[22] V. Veeriah, N. Zhuang, and G. Qi. Differential recurrent neural networks for action recognition. In *ICCV*, pages 4041–4049, 2015.

[23] R. Vemulapalli, F. Arrate, and R. Chellappa. Human action recognition by representing 3D skeletons as points in a Lie group. In *CVPR*, pages 588–595, 2014.

[24] R. Vemulapalli and R. Chellappa. Rolling rotations for recognizing human actions from 3D skeletal data. In *CVPR*, pages 4471–4479, 2016.

[25] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *CVPR*, pages 1290–1297, 2012.

[26] D. Wu and L. Shao. Leveraging hierarchical parametric networks for skeletal joints based action segmentation and recognition. In *CVPR*, pages 724–731, 2014.

[27] L. Xia, C. Chen, and J. K. Aggarwal. View invariant human action recognition using histograms of 3D joints. In *CVPRW*, pages 20–27, 2012.

[28] G. Yu, Z. Liu, and J. Yuan. Discriminative orderlet mining for real-time recognition of human-object interaction. In *ACCV*, pages 50–65, 2014.

[29] M. Zanfir, M. Leordeanu, and C. Sminchisescu. The moving pose: An efficient 3D kinematics descriptor for low-latency action recognition and detection. In *ICCV*, pages 2752–2759, 2013.

[30] X. Zhao, X. Li, C. Pang, X. Zhu, and Q. Z. Sheng. Online human gesture recognition from motion data streams. In *ACM MM*, pages 23–32, 2013.

[31] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, and X. Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. In *AAAI*, pages 3697–3704, 2016.