RECOGNIZING PLANS BY LEARNING EMBEDDINGS FROM OBSERVED ACTION DISTRIBUTIONS

HANDLING UNCERTAINTY IN VISUAL PERCEPTION FOR PLAN RECOGNITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Plan recognition aims to look for target plans to best explain the observed actions based on plan libraries and/or domain models. Despite the success of previous approaches on plan recognition, they mostly rely on correct action observations. Recent advances in visual activity recognition have the potential of enabling applications such as automated video surveillance. Effective approaches for such problems would require the ability to recognize the plans of agents from video information. Traditional plan recognition algorithms rely on access to detailed planning domain models. One recent promising direction involves learning approximate (or shallow) domain models directly from the observed activity sequences. Such plan recognition approaches expect observed action sequences as inputs. However, visual inference results are often noisy and uncertain, typically represented as a distribution over possible actions. In this work, we develop a visual plan recognition framework that recognizes plans with an approximate domain model learned from uncertain visual data.

1 INTRODUCTION

Plan recognition aims to look for target plans to best explain the observed actions based on plan libraries (Kautz and Allen (1986)) and/or domain models (Ramírez and Geffner (2009); Zhuo, Yang, and Kambhampati (2012)). Despite the success of previous approaches proposed to handle plan recognition problems, they generally rely on the assumption that the observed actions are correct and try to discover missing actions (c.f. Tian, Zhuo, and Kambhampati (2016)). Even though there have been previous works on noisy observations, they only consider the case that the observed actions (c.f. Zhuo and Kambhampati (2013)) or states (c.f. Mourão et al. (2012)) have a probability to be incorrect, instead of considering full distribution of actions (or states). In many real-world applications, such as video surveillance (c.f. Collins et al. (2000)), a full distribution of actions, which can be directly estimated by off-the-shelf approaches (e.g., Zhong, Shi, and Visontai (2004)), is however available and can be explored to help recognize plans.

Consider the example in Figure 1, where a surveillance system attempts to recognize plans from captured video clips of salad-making activities. In the first two frames, the activity recognition module may generate a distribution over three activities, and determine that the human grasps a pestle with the highest confidence (probability), a cucumber and a pen with lower confidences. In such a case there is a perceptual error (PE), assuming the ground-truth activity of picking up a cucumber. If the plan recognition module only takes the most likely activity (i.e. grasping a pestle), only erroneous information is used by the plan recognition module, leading to a poor performance. Hence the plan recognition module should consider the information of less likely activities as well.

In this work, we consider the problem of recognizing plans from videos, namely visual plan recognition, based on the distribution of actions at each step of the underlying plan to be discovered. Visual plan recognition is challenging due to the complex affinities of candidate actions among multiple steps of the underlying plan based on distributions of actions. In this paper we propose to learn action embeddings to capture affinities according to the distributions of actions, and then exploit the action embeddings to recognize plans.

Specifically, to learn action embeddings from distributions of actions, we propose three candidate approaches. The first one is to greedily take the most probable action from each distribution, namely

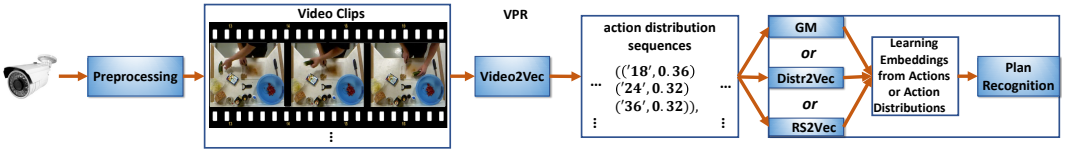

Figure 1: This figure illustrates the framework of VPR, which does plan recognition on uncertain observations. Those uncertain observations come from a visual recognition model, Video2Vec, whose training is explained in the appendix (Section B). The number '18', '24', and '36' means the 18-th, 24-th, and 36-th action in action vocabulary $\bar{A}$ (explained in Section 3)

GM (Greedy Model). GM may suffer from noisy observations because of greedy sampling. The second, in order to use more information from uncertain observations, is to sample action sequences from each sequence of action distributions, and then build a Word2Vec model, namely RS2Vec (Resampling-to-Vector Model). RS2Vec can be quite inefficient in that it will have to generate many samples to capture the distribution. The required sample size increases with the length of the sequence and the number of actions in the distribution of each step. We thus propose the third way, which is to directly handle distributions of actions, namely Distr2Vec (Distribution-to-Vector model). Distr2Vec requires a revision to the Word2Vec to let it learn embeddings for distributions over words as opposed to single words. To train the Distr2Vec affinity model, we introduce a loss function based on combining Kullback-Leibler (KL) divergence and (Hierarchical-Softmax) H-Softmax Mikolov et al. (2013).

The framework, namely VPR (visual plan recognition), is shown in Figure 1. A pretrained action recognition module converts a sequence of video clips from a camera, to a sequence of observed action distributions (i.e., observed plans). Then we could do plan recognition by using one of the three affinity models, GM, RS2Vec, or Distr2Vec. Those affinity models serve as approximated planning domain models and generate affinity scores for plan recognition. An affinity score represents how likely an action is going to happen given observed actions.

To the best of our knowledge, we are the first to learn approximated domain (action affinities) models from sequences of action distributions that are gathered from an activity recognition model trained on real-world videos. Further, the potential of Distr2Vec and RS2Vec is not restricted to visual plan recognition. They may also be useful in any categorical data sequence learning scenario where there is uncertainty at each step of the input data such as in speech recognition.

2 Related Work

In the planning literature, there has been a long history of leveraging models for plan recognition *but without considering connections to low-level perception*. In Sohrabi, Riabov, and Udrea (2016) and Ramírez and Geffner (2009), solving a plan recognition problem is transformed to solving a planning problem. These two plan recognition works assume that a planning domain model is given, whose description strictly follows certain rules or syntaxes, like those in PDDL McDermott et al. (1998). Recently an interesting direction that has emerged, is to learn an approximated planning model. These models could be flexibly represented in various formats, to do plan recognition, as in Tian, Zhuo, and Kambhampati (2016), which exploits Word2Vec to learn approximated planning models from plan corpora. In contrast to these works, our VPR can handle uncertain information in action recognition from a visual processing model. Our work is also different from existing works of probabilistic plan recognition (Bui (2003); Ramírez and Geffner (2010); Geib and Goldman (2009)). These works require a predefined set of rules or a domain model, which is instead approximately learned from plan traces in our VPR.

Another relevant body of research is intention recognition from visual observations (Xie et al. (2018b); Kitani et al. (2012); Holtzen et al. (2016)). The intention recognition modules in these works remove observation uncertainty by taking only the most likely observed (or demonstrated) trajectories and object states from a tracking module in a similar sense of our GM. They will therefore lose the information in the distribution-style observation sequences. Our work may be used to

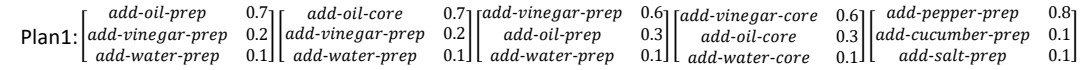

Figure 2: An uncertain plan p, which is a sequence of observed action distributions.

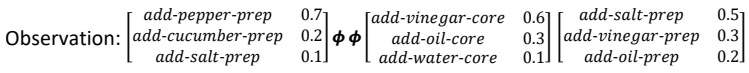

Figure 3: An observed plan O, that has two steps of missing observations (action distributions) marked with symbol ϕ.

augment these and similar visual intention recognition works because our approach preserves the information from the uncertainty in the data.

Using sensor data as input, Hodges and Pollack designed machine learning-based systems Hodges and Pollack (2007) for identifying individuals as they perform routine daily activities such as making coffee. Liao et al. proposed to infer user transportation modes Liao et al. (2007) from readings of radio-frequency identifiers (RFID) and global positioning systems (GPS). Freedman et al. explore the application of natural language processing (NLP) techniques Freedman, Jung, and Zilberstein (2014), i.e., Latent Dirichlet Allocation topic models, to human skeletal data of plan execution traces obtained from a RGB-D sensor. Bulling et al. discussed the key research challenges Bulling, Blanke, and Schiele (2014) that human activity recognition shared with general pattern recognition. When activity recognition is performed indoors and in cities using the widely available Wi-Fi signals, there is much noise and uncertainty. Xie et al. proposed a temporal-then-spatial recalibration scheme to build an end-to-end Memory Attention Networks (MANs) Xie et al. (2018a) for solving skeleton-based action recognition task. Chen et al. propose a multi-agent spatial-temporal attention model Chen et al. (2019) to jointly recognize activities of multiple agents. To tackle the limitations of feature extraction and training data labeling effort, Qian et al. propose a distribution based semi-supervised learning approach Qian, Pan, and Miao (2019) to recognize human activities. Different from the above-mentioned sensor-based activity recognition, we aim to recognize plans, i.e., sequences of actions, instead of predicting single activities.

3 Visual Plan Recognition

Definition 1. *(Action Space)* We define the action space as $A = \bar{A} \cup \phi$. The action set A consists of all possible grounded action symbols (in action vocabulary $\bar{A}$), and a symbol ϕ which denotes a step with missing observation (action distribution).

Definition 2. *(Action Distribution)* An observed action distribution of size K at time step t is denoted as $Distr(a_t)$ which equals to $\langle (a_t^1, c_t^1), (a_t^2, c_t^2), ..., (a_t^K, c_t^K) \rangle$. Note that each action a has a corresponding confidence value c. c is a probability given by a visual recognition module, and $\sum_k^K c_k = 1$.

Definition 3. *(Uncertain Plan)* We use p to define an observed plan with uncertainty, and T to denote its length or number of steps. A plan p with T steps would be a sequence of T action distributions, defined as the matrix below.

$$\text{Actions} \downarrow \quad \overrightarrow{\text{Time}} \quad \begin{pmatrix} a_1^1, c_1^1 & a_2^1, c_2^1 & a_3^1, c_3^1 & \cdots & a_T^1, c_T^1 \\ a_1^2, c_1^2 & a_2^2, c_2^2 & a_3^2, c_3^2 & \cdots & a_T^2, c_T^2 \\ a_1^3, c_1^3 & a_2^3, c_2^3 & a_3^3, c_3^3 & \cdots & a_T^3, c_T^3 \\ \vdots & \vdots & \vdots & \ddots & \cdots \\ a_1^K, c_1^K & a_2^K, c_2^K & a_3^K, c_3^K & \cdots & a_T^K, c_T^K \end{pmatrix}$$

Definition 4. *(Visual Plan Recognition)* The visual plan recognition problem is defined as $R = (L, O, A)$. L is a library of uncertain plan traces p (Figure 2). A denotes the action space as explained. O is an observed plan captured by running visual activity recognition modules on video

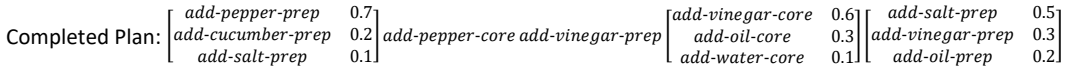

Figure 4: The completed plan after running visual plan recognition on observed plan O.

inputs. O may contain missing observations o_t at certain steps t (Figure 3). To recognize a plan we need to search for proper actions to fill them in those steps. The searching requires using action affinity models, whose training will be explained later. Action affinity model works as a subroutine in VPR. The solution R, as illustrated in Figure 4, could be either a completed plan with all missing actions filled in (plan completion), or a plan that includes future actions of an agent (plan recognition). Note that plan completion is a more general version of plan recognition problem such that missing observation can be at any step in a plan. For plan recognition, missing observations are only at the end of a plan. In some earlier plan recognition works Ramírez and Geffner (2009); Zhuo, Yang, and Kambhampati (2012) they did solve plan completion problems as well, as pointed out by Tian, Zhuo, and Kambhampati (2016). Similar to their works, we also mean to solve the more general problem of plan completion and assume that the missing observations can be at any plan step.

3.1 Formulation of Learning Action Affinity Models

We can now formulate the learning of action affinity model, as maximizing the mean of log probability of distributions over the steps of plans in L:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-\mathcal{W}\leq j\leq\mathcal{W},j\neq 0}\log Pr(Distr(a_{t+j})|Distr(a_t)) \tag{1}$$

where $\mathcal{W}$ denotes the window size. $Distr(a_t)$ denotes the input observation distribution ($input$) at the current step. $Distr(a_{t+j})$ denotes the target observation distribution ($target$, which is the expected output) at step $t+j$, which provides training signals. $Pr(Distr(a_{t+j})|Distr(a_t))$ represents the similarity between $Distr(a_{t+j})$ and $Distr(a_t)$, computed by an affinity model.

Also, when using a $Distr(a_t)$, the affinity model encodes $Distr(a_t)$ into a vector $enc(Distr(a_t)) = u_1, ..., u_{\bar{A}}$. The size of $\bar{A}$ equals to the number of nodes in input layer. $Distr(a_t)$ is encoded into a vector by having a unique index i associated to each action in $\bar{A}$. The value u_i at each index i in the input vector, is the confidence value associated to the matching action in $Distr(a_t)$. The rest of the units in input layer, whose corresponding actions are not in $Distr(a_t)$, have zero probability values.

3.2 Recognizing Plans with a Learned Affinity Model

For an observed plan O (**Definition 4.**), if there are totally M steps or observations $(o_1, ..., o_M)$, the length of O is M. If there are x steps in O with missing observed action distributions, there are x ϕ symbols in O (**Definition 1.**). To recognize the plan O we try different actions within $\bar{A}$ to fill in each of the x steps, and select the actions that maximize overall affinities. Note that x is a variable that may vary plan by plan.

For example, consider when $x = 1$ that there is only one missing observation (at step j) in the observed plan O. All observations at other steps are known (action distributions). To determine the action for a step of missing observation, we calculate $\bar{A} \times 2\mathcal{W}$ pairwise affinities for all possible actions that could complete O. $\mathcal{W}$ is the context window size (a hyperparameter) assumed to be set to one in this example. Each candidate plan completion is denoted as $\widetilde{p}$. $\widetilde{p}$ is also a M-step plan but with symbols ϕ replaced by actions in $\bar{A}$. To score $\widetilde{p}$ with one filled-in action in our example, we need to compute the pairwise affinities between all possible actions in $\bar{A}$, and an observed action distribution within $-\mathcal{W} \leq j \leq \mathcal{W}$. The selected action would have the highest pairwise affinity to observations in the window. As a result, because the plan only has one step with missing observation, the selected action completes the $\widetilde{p}$ and we obtain a recognized plan.

The score of a plan completion $\widetilde{p}$, denoted as $\mathcal{F}(\widetilde{p})$, is calculated using the sum of log probabilities $Pr(target|input)$ (measuring a pairwise affinity) over all M steps:

$$\mathcal{F}(\widetilde{p}) = \sum_{t=1}^{M} \sum_{-\mathcal{W} \leq j \leq \mathcal{W}, j \neq 0} \log Pr(target|input) \quad (2)$$

where both *input* and *target* are *encodings* of either a single action (a one-hot vector), enumerated from the action vocabulary, that could be put in a step with missing observation, or an observed action distribution (e.g., $enc(Distr(a_t))$) in the window $\mathcal{W}$. Note that *target* action or action distribution is the expected output. Each $\widetilde{p}$ (has length M) is a candidate plan completion, in which x actions would be filled in the x steps with missing observations in an observed plan O (**Definition 4.**). j is the index inside context window $\mathcal{W}$. The plan completion for $\widetilde{p}$ that has the highest score, is treated as the solution R for visual plan recognition. The general procedure is described in **Algorithm. 1**.

Algorithm 1 Plan Recognition with an Affinity Model

INPUT: An observed plan O, and an affinity model $\mathcal{M}^{\mathcal{A}}$
OUTPUT: A plan completion $\widetilde{p}$

```
for i = 1; i ≤ M; i++ do
    for steps in −W ≤ i ≤ W do
        if any step j has the value φ then
            for each action a from Ā do
                φ ←a                    ▷ Replace φ with a
                Enumerate input-target pairs and compute their affinity values with M^A
                Store the action a and corresponding affinity value
            end for
        end if
    end for
end for
Select actions with maximum affinity values to form p̃
return p̃
```

4 Action Affinity Models

In this section, we formally introduce our action affinity models, `Distr2Vec` and `RS2Vec`, that could be used in `VPR` framework for visual plan recognition.

4.1 Distrbution-to-Vector (Distr2Vec)

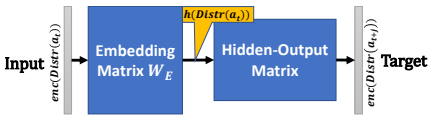

Figure 5: Our `Distr2Vec` for learning action embeddings (approximated planning domain models) directly from distribution sequences. The embedding matrix W_E has the size of action vocabulary $\bar{A}$ times embedding size. The hidden-Output Matrix has the shape of embedding length times size of $\bar{A}$, whose output is a vector of size $\bar{A}$.

Figure 5 illustrates our `Distr2Vec` model, that takes a pair of action distribution encodings to update its embeddings. The pair of distributions are enumerated within the context window. `Distr2Vec` takes one as $input$ and treats the other as the $target$. Each row in W_E is the embedding of an action symbol.

We train `Distr2Vec` by minimizing the KL-divergence between the target distribution $Distr(a_{t+j})$, and the predicted output distribution $\hat{Distr}(a_{t+j})$. KL-divergence is an often-used measure of the distance between two distributions. So, if we use it as the loss function we can keep the input data as is (no resampling). Using KL-divergence as the loss function, `Distr2Vec` is trained by minimizing:

$$D_{KL}(enc(Distr(a_{t+j}))||enc(\hat{Distr}(a_{t+j}))) \quad (3)$$

where D_{KL} represents the KL divergence. $enc(Distr(a_t))$ are defined in Section. 3. KL-divergence D_{KL} is calculated in Equation 4.

$$KL(p||q) \triangleq \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k} = \sum_{k=1}^{K} p_k \log p_k - \sum_{k=1}^{K} p_k \log q_k \quad (4)$$

where q is the output probability distribution of our `Distr2Vec` model. q is also an approximation of p, the target distribution $Distr(a_{t+j})$ in the plan trace. Equation 4 allows us to avoid computing the derivative of the entropy of p when taking partial derivative of $KL(p||q)$ with respect to model parameters. This is because the probability values for p (which is $Distr(a_{t+j})$), are constants with

respect to the model parameters. Using this information, we can obtain Equation 5.

$$\begin{aligned} & D_{KL}(enc(Distr(a_{t+j}))||enc(\hat{Distr}(a_{t+j}))) \\ &= Z(Distr(a_{t+j})) - \sum_{k=1}^{K} c_{t+j}^{k} \log Pr(a_{t+j}^{k}|h(Distr(a_t))) \end{aligned} \tag{5}$$

where $Z(Distr(a_{t+j})) = \sum_{k=1}^{K} c_{t+j}^{k} \log(c_{t+j}^{k})$ is a constant, and $h(Distr(a_t))$ is the embedding computed by multiplying the embedding matrix W_E and the action-distribution input vector $enc(Distr(a_t)) = \langle u_1, u_2, ..., u_{\bar{A}} \rangle = \langle 0...0, c_t^1, 0, ..., 0, c_t^2, 0, ...0, c_t^K, 0... \rangle$ encoded from $Distr(a_t)$, as done in Equation 6. u, $\bar{A}$,c_t, and K are defined in Section. 3.

$$h(Distr(a_t)) = W_E \times enc(Distr(a_t)) \tag{6}$$

4.2 Combining with Hierarchical Soft-max

We adopted the H-Softmax introduced in Mikolov et al. (2013), which has been shown to have advantages over the non-hierarchical counterpart in both the accuracy, and computational efficiency[1]. H-Softmax decomposes the logits computation of each word to computing a sequence of nodes (logit values) in a tree data structure.

If we combine Equation 5 with H-Softmax, with distribution input $Distr(a_t)$, we obtain the probability of an action a_{t+j}^k in the target observed action distribution $Distr(a_{t+j})$:

$$\begin{aligned} Pr(a_{t+j}^{k}|h(Distr(a_t))) = & \prod_{i=1}^{L(a_{t+j}^{k})-1} \Big\{ \sigma(\mathbb{I}(n(a_{t+j}^{k}, i+1) \\ & = child(n(a_{t+j}^{k}, i))) \cdot v_{n(a_{t+j}^{k}, i)} \cdot h(Distr(a_t))) \Big\} \end{aligned} \tag{7}$$

where $\mathbb{I}(x)$ is an identity function. $L(a_{t+j}^k)$ is the length of path from root to the leaf node, i.e., an action, in a H-Softmax tree. $v_{n(a_{t+j}^k, i)}$ is the weights vector of ith node along the path in the tree, and h is the embedding computed using Equation 6. We encourage readers to read a brief review of the `Word2Vec` and H-Softmax in the Section D of appendix, to understand Equation 7 better.

And if we substitute Equation 7 into Equation 5, we obtain the error function in Equation 8. This is the error function as we are trying to minimize the KL divergence of Equation 5.

$$E = Z(Distr(a_{t+j})) - \sum_{k=1}^{K} c_{t+j}^{k} \sum_{i=1}^{L(a_{t+j}^{k})-1} \Big\{ \log \sigma(\mathbb{I}(.) v_{n(a_{t+j}^{k}, i)} \cdot h(Distr(a_t))) \Big\} \tag{8}$$

Note that we also use Equation 8 to calculate the logarithmic posterior probability of obtaining target encoding given input encoding, $\log Pr(target|input)$, as in Equation 2. Readers should refer to the Section E in appendix for detailed derivation of gradient descent for Equation 8.

4.3 Resampling-to-Vector (RS2Vec)

In `RS2Vec`, we first perform a beam search sampling on each uncertain plan p from library L, to obtain a set of deterministic samples (plans). Each sample includes a sequence of actions, and a probability value calculated by multiplying the confidence values of all actions along a sequence. This probability value could be used to represent the overall uncertainty or score of a sampled plan. After doing beam search sampling in which the beam size is set to S, the top S samples

[1]The first version that tried to improve Word2Vec with H-Softmax in Morin and Bengio (2005), reports that the new model requires less training time, but also results in a degraded accuracy. However, the work Mnih and Hinton (2009) introduces an approach to automatically grow a tree to organize words, for H-Softmax, which outperforms non-hierarchical Word2Vec. In the recent work Mikolov et al. (2013), the advantage is confirmed and it selects the binary Huffman tree as the basic tree data structure.

(action sequences) with their scores are selected. We then resample them by using the roulette wheel resampling approach of Lipowski and Lipowska (2012), which lets us drop some samples that have low scores, and increase the amount of samples with high scores. After doing resampling we use the action sequences in all stored samples after performing beam search, to train a `Word2Vec` model as an action affinity model for plan recognition.

That said, the potential problem for `RS2Vec` are the following. First, the resampling would make the algorithm more computationally expensive, and the training time would be closely related to the number of samples and length of the training data. This makes it hard to apply `RS2Vec` to a real-time plan recognition system. Secondly, the resampling approach could lose some information, whereas `Distr2Vec` directly uses the entire distribution sequence.

5 EVALUATION

We evaluate `Distr2Vec` and `RS2Vec` in `VPR` by observing training time and the plan recognition performance of `VPR`. `VPR` takes `Distr2Vec`, `RS2Vec`, and the baseline model `GM`. In `GM`, we take sequences of most likely actions from distributions across each plan to build a plan corpus. With the corpus we train an action affinity (`Word2Vec`) model. This is as opposed to the `RS2Vec` model which takes more samples per sequence for training. We ran our experiments on a machine with a Quad-Core CPU (Intel Xeon 3.4GHz), a 64GB RAM, a GeForce GTX 1080 GPU, and Ubuntu 16.04 OS.

5.1 DATASET COLLECTION AND EXPERIMENT DESIGN

We collected two plan corpora based on processing the 50 Salads Dataset Stein and McKenna (2013). One is from real world videos (visually grounded corpus), for evaluating the real world value of our model in connecting high-level plan recognition and low-level perception. The other corpus is a controlled synthetic plan corpus. In order to fully assess the validity and effectiveness of our approaches, we need to test with different types of distributions in input data. Thus we generated synthetic plans (distribution sequences), as they would allow us to maximally evaluate the approach by testing the effects of various factors. We evaluate the effect of factors like distribution entropies, perception error rates (PER), and the length of observation sequences. Note that we simulate the perceptual error (PE) in an action distribution by exchanging the highest confidence action with another action in that distribution. A specified PER is achieved by simulating PE in a proportion of the action distributions in each plan trace. For experiments on synthetic data we evaluate affinity models with H-Softmax. We also explore the performance difference from using their counterparts without H-Softmax in terms of training time and accuracy in Section 5.3. We set the size of window $\mathcal{W}$ to one, embedding size to 100 and train each model for 60 epochs. For each position of missing observation, we ask models to predict top-three actions in all experiments except for Figure. 8. The prediction is counted as correct if at least one of the three is correct. We denote the number of candidate predictions as top-k-prediction. For details of dataset collection and experiment settings, readers should refer to section B and C in the appendix.

We evaluate different approximated domain models (`Distr2Vec`, `RS2Vec`, and `GM`) as used in `VPR` using 6-fold cross validation. For each test, we randomly remove some distributions at some steps. Then we compare the performance by the average accuracy. We define the accuracy following the work Tian, Zhuo, and Kambhampati (2016), using Equation 9.

$$acc = \frac{1}{Z}\sum_{i=1}^{Z}\frac{\#\langle CorrectRecognition\rangle_i}{x_i} \tag{9}$$

Where Z is the total number of plan traces in testing set, and x_i is the number of steps with missing actions for the plan i. $\#\langle CorrectRecognition\rangle_i$ is the number of correct predicted actions in i-th uncertain plan in testing set.

5.2 EXPERIMENTS AND ANALYSIS ON SYNTHETIC PLANS

Figure 6 shows accuracy results for synthetic plans of length 10, 20 and 30, with varying PER and fixed entropy, in the four sub-plots on the left. The `RS2Vec`-S, `RS2Vec`-M, and `RS2Vec`-L are

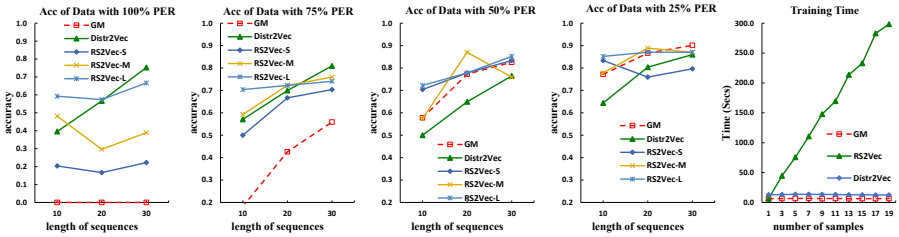

Figure 6: This figure demonstrates the accuracy of GM, Distr2Vec, and three RS2Vec instances on synthetic data of different **PERs**, with respect to increasing plan (distribution sequence) lengths. The three RS2Vecs (RS2Vec-S, RS2Vec-M, and RS2Vec-L), are RS2Vec instances that are trained on small (S), medium (M), and large (L) numbers of samples.

three RS2Vec instances trained on three levels of samples (small, medium, and large). We set the sample numbers of small, medium, and large levels to 3, 9, and 27 respectively. In the rightmost sub-plot, we show a comparison of **training time** required by GM, Distr2Vec, and RS2Vec with respect to increasing number of samples that an RS2Vec instance takes, when sequence length is 20. We did training time experiments for length 10, 20, and 30 of sequences, but because the three training time results are very similar to each other, we only report the length 20 results.

We observe that the training time of RS2Vec increases linearly with the number of samples per plan. This is easily explained by the fact that increasing the number of samples, is increasing the training cost. In contrast, the training time for Distr2Vec and GM stay constant as there is no sampling step.

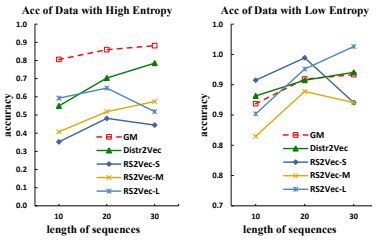

Figure 7: Accuracy results of GM, Distr2Vec, and three RS2Vec instances on synthetic data of different **entropies**. The RS2Vec-S, RS2Vec-M, and RS2Vec-L are explained in the description of Figure 6.

For accuracy performances under all PER settings, Distr2Vec generally gets better performances for longer training sequences. As the length of plan increases, RS2Vec requires more samples from the distribution sequence to match or outperform the Distr2Vec (this is more obvious when PER is high). For RS2Vec, increasing the number of samples could be beneficial when PER is higher. Higher PER means more observed action distributions in a plan would have lower probabilities for correct actions (assuming each distribution must have one action that matches the ground-truth). Thus, taking more samples would give RS2Vec more accesses to correct information. We can observe clearly in sub-plots of 100% and 75% PER, that RS2Vec could get better performance by training on more samples.

As for the GM model, its accuracy for 100% PER across all lengths is 0%. This matches our expectation sicne the training data for the GM model does not have the ground-truth action in any step for 100% PER. Therefore, the GM would not capture any relevant information. For lower PER cases the GM performance is higher. This matches our expectations since the GM gets more of the correct information from highest-probability sampling of low PER plans.

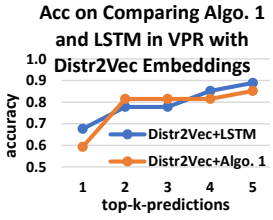

Figure 8: Algorithm. 1 and LSTM comparison in VPR framework for plan recognition.

As for data with high and low entropies, the experimental results are shown in Figure. 7. We set PER to zero, and sample numbers of RS2Vec-S, RS2Vec-M, and RS2Vec-L, to 7, 11, and 15 respectively. Then the actions which match ground-truth ones are most probable ones in all distributions. As expected, GM performs the best when entropy of distributions in training distribution sequences (plan corpus) is high, since it gets the correct plan when PER is zero. The performance of Distr2Vec is expectedly lower than GM, yet it gets appreciably bet-

ter than RS2Vec. In low entropy cases all models perform comparably well.

We also evaluated Algorithm 1 in VPR (Section 3.2) by comparing it with the Long-shot Term Memory networks (LSTM) approach Graves (2013). Our LSTM based approach is essentially a generative model, implemented based on [2]. It first learns action representations based on Distr2Vec and then uses the learnt representations to train the LSTM for predicting an action. Different from Algorithm 1, LSTM can do plan recognition by approximating an optimal prediction, which may require a slower training time but do faster in testing (plan recognition) phase. We would like to investigate the differences in accuracy and running time between the two approaches. The accuracy result is shown in Figure. 8 and we used the synthetic data of 30% PER. The VPR with Algorithm. 1 and VPR with LSTM performs comparably well with top-k-prediction value increases. In addition, VPR with LSTM requires 29.5 Secs to train and 0.02 Secs to recognize one action in testing phase. VPR with Algorithm 1 requires 8.0 Secs and 0.80 Secs respectively. Hence VPR with Algorithm 1 overall has a less running cost.

5.3 Experiments and Analysis on visually grounded Plans

In Figure 9, the left sub-plot shows how Distr2Vec, GM, and RS2Vec which takes 9 (as RS2Vec-S) and 27 (as RS2Vec-L) samples would perform, when PER is high (69%) and low (8%), meanwhile setting distribution size to 5. The right sub-plot shows how sizes of distributions per step influence the accuracy (setting RS2Vec to takes 15 samples). We observe that, when PER is high, VPR with Distr2Vec clearly performs the best, whereas when PER is low, all models perform comparably well. This is consistent to results of synthetic data evaluation. We also observe that, overall the distribution size is not a key factor for deciding model performance.

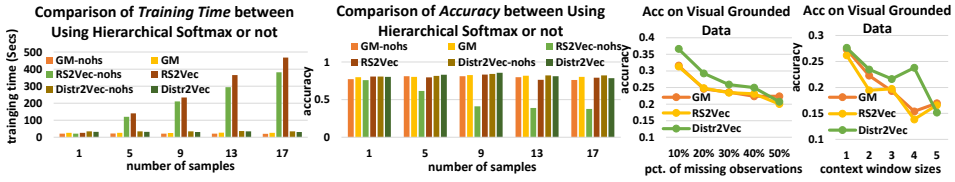

Figure 10: The left two subplots show a comparison between using H-Softmax or not with respect to training time and accuracy for affinity models on the visually grounded corpus of 8% PER. The suffix "-nohs" denotes models without H-Softmax. The right two subplots show accuracy with respect to the percentage of missing observations and window sizes on the visually grounded corpus of 69% PER.

The left two subplots in Figure 10 show that the H-Softmax in Distr2Vec slightly decreases the training time, and vice versa for GM. This is probably because our action space is small (52). H-Softmax is faster for corpora with larger vocabularies as the tree traversal way of updating weights will eventually take less time than updating the entire hidden-output matrix. The accuracy advantage of using H-Softmax is obvious for RS2Vec with more samples. The reason is RS2Vec-nohs with more samples tends to sample more incorrect data. RS2Vec tends to be more robust to this noise. From the right two subplots we observe that the optimal window size is one, and 50% of missing observations per plan is the lower bound for Distr2Vec to maintain its advantage over other models.

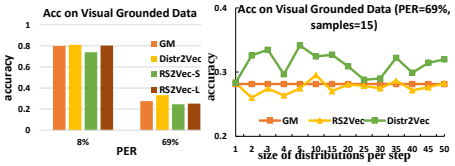

Figure 9: Results of evaluating the GM, RS2Vec, and Distr2Vec on videos. The left sub-plot shows how Distr2Vec, GM, and RS2Vec with small and large number of samples (RS2Vec-S and RS2Vec-L) performs with data of 8% and 69% PERs. The right subplot shows the accuracy in terms of varying distribution sizes on the 69% PER data.

[2] https://github.com/pytorch/examples/tree/master/word_language_model

6 CONCLUSION AND FUTURE WORK

Our work aims to make plan recognition more practically useful in scenarios with observational error in action recognition. We introduce our VPR framework that uses Distr2Vec and RS2Vec as action affinity models, to do visual plan recognition which may suffer from perceptual uncertainty. Distr2Vec and RS2Vec learn embeddings for distributions over actions to preserve the information from the visual recognition module. Our VPR framework can thus handle the uncertainty in visual data and perform plan recognition. We evaluated our models (Distr2Vec and RS2Vec) as well as a baseline GM model in the context of plan recognition in the VPR framework. We evaluated models with/without H-Softmax on plans in which action distributions are generated from real-world videos. We also evaluated the H-Softmax version of models on synthetic plans with different types of action distributions. We conclude that when there is a higher PER, VPR with Distr2Vec outperforms other models. When PER is lower, and sequence length is longer, all models perform comparably well. Another advantage of using Distr2Vec to handle distributions within the VPR framework is that the training time is shorter as compared to the RS2Vec model in which we have to sample more action sequences for comparable performance. For future work we would be interested in exploring other domains that provide uncertain observations, like speech recognition.

REFERENCES

Reh u rek, R., and Sojka, P. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50. Valletta, Malta: ELRA.

Bui, H. H. 2003. A general model for online probabilistic plan recognition. In *IJCAI*, volume 3, 1309–1315. Citeseer.

Bulling, A.; Blanke, U.; and Schiele, B. 2014. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Comput. Surv.* 46(3):33.

Chen, K.; Yao, L.; Zhang, D.; Guo, B.; and Yu, Z. 2019. Multi-agent attentional activity recognition. *CoRR* abs/1905.08948.

Collins, R. T.; Biernacki, C.; Celeux, G.; Lipton, A. J.; Govaert, G.; and Kanade, T. 2000. Introduction to the special section on video surveillance. *IEEE Trans. Pattern Anal. Mach. Intell.* 22(8):745–746.

Freedman, R. G.; Jung, H.-T.; and Zilberstein, S. 2014. Plan and activity recognition from a topic modeling perspective. In *Proceedings of ICAPS*.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101–1132.

Graves, A. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hodges, M. R., and Pollack, M. E. 2007. An 'object-use fingerprint': The use of electronic sensors for human identification. In *Ubicomp*, 289–303.

Holtzen, S.; Zhao, Y.; Gao, T.; Tenenbaum, J. B.; and Zhu, S.-C. 2016. Inferring human intent from video by sampling hierarchical plans. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, 1489–1496. IEEE.

Hu, S.-H.; Li, Y.; and Li, B. 2016. Video2vec: Learning semantic spatial-temporal embeddings for video representation.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of AAAI*, 32–37.

Kitani, K. M.; Ziebart, B. D.; Bagnell, J. A.; and Hebert, M. 2012. Activity forecasting. In *European Conference on Computer Vision*, 201–214. Springer.

Liao, L.; Patterson, D. J.; Fox, D.; and Kautz, H. A. 2007. Learning and inferring transportation routines. *Artif. Intell.* 171(5-6):311–331.

Lipowski, A., and Lipowska, D. 2012. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications* 391(6):2193–2196.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Mnih, A., and Hinton, G. E. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, 1081–1088.

Morin, F., and Bengio, Y. 2005. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, 246–252. Citeseer.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS operators from noisy and incomplete observations. In *UAI*, 614–623.

Qian, H.; Pan, S. J.; and Miao, C. 2019. Distribution-based semi-supervised learning for activity recognition. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, 7699–7706.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1778–1783.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *IJCAI*, 3258–3264.

Stein, S., and McKenna, S. J. 2013. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2013), Zurich, Switzerland*. ACM.

Tian, X.; Zhuo, H. H.; and Kambhampati, S. 2016. Discovering underlying plans based on distributed representations of actions. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 1135–1143. International Foundation for Autonomous Agents and Multiagent Systems.

Xie, C.; Li, C.; Zhang, B.; Chen, C.; Han, J.; and Liu, J. 2018a. Memory attention networks for skeleton-based action recognition. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 1639–1645.

Xie, D.; Shu, T.; Todorovic, S.; and Zhu, S.-C. 2018b. Learning and inferring "dark matter" and predicting human intents and trajectories in videos. *IEEE transactions on pattern analysis and machine intelligence* 40(7):1639–1652.

Zhong, H.; Shi, J.; and Visontai, M. 2004. Detecting unusual activity in video. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA*, 819–826.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *IJCAI*, 2444–2450.

Zhuo, H. H.; Yang, Q.; and Kambhampati, S. 2012. Action-model based multi-agent plan recognition. In *Advances in Neural Information Processing Systems*, 368–376.

A APPENDIX

B DATASET COLLECTION

B.1 COLLECTION OF VISUALLY GROUNDED PLAN CORPUS

The procedure for making the visually grounded corpus is illustrated in Figure 1. First, we converted all videos into video clips, and each clip matches a ground-truth action. Secondly, we applied a video processing model, Video2Vec Hu, Li, and Li (2016), on those video clips, to output action distribution sequences. For each clip it outputs a distribution over all actions. We concatenate distribution sequences together for each video to make plan corpora that correspond to real-world videos. We trained two visual recognition models. The first is trained on 90% of video clips with 450 epochs. The second one is trained on 70% of video clips with 100 epochs. With the two visual models trained by Video2Vec, we obtain two plan corpora. We found the PER for plan corpora collected by running the first visual model, is (8%), and the PER for plans collected by running the second visual model is 79%.

B.2 COLLECTION OF SYNTHETIC PLAN CORPUS

We also augmented the ground-truth sequences in 50 Salads Dataset Stein and McKenna (2013) to create a synthetic plan corpus of action distribution sequences. There are totally 54 ground-truth, low-level action sequences, $a^*_{1:T}$ (T is sequence length). We synthesize a distribution per action step, by adding actions to each step and assigning a probability distribution over the actions. We add $K-1$ additional actions to each observation and assign a probability distribution such that the ground truth has the highest confidence (probability).

In order to generate the $K-1$ additional actions, we search for the $K-1$ most similar actions by using a `Word2Vec` model that is pretrained on Google News corpus [3]. We use the semantic correlation of actions in this model, to simulate their visual correlation. As for assigning confidence values for each of the $K-1$ additional actions, we followed the Equation 10.

$$c(a^k_t) = \frac{s_k}{1 + w_{entropy} + \sum_{i=1}^{K-1} s_i} \tag{10}$$

where a^k_t is the k^{th} additional action in a distribution at step t. We search for a^k_t in the `Word2Vec` model based on the ground truth action a^*_t. s_k is the similarity between a^*_t, and a^k_t, which can be computed by using the pretrained `Word2Vec` of gensim library Reh u rek and Sojka (2010). These similarity values are used as in the Equation 10. The $+1$ in the denominator denotes the similarity between the ground-truth action and itself. We also use a parameter $w_{entropy}$ which we can use to increase or decrease probability differences between actions. Thus it could be used to adjust the entropy of each action distribution. We set up $w_{entropy}$ for the computation of each $c(a^k_t)$ to either zero or one, to slightly increase the variance of confidences at each step. The confidence for ground-truth action a^*_t is initially the highest and is equal to Equation 11.

$$c(a^*_t) = 1 - \sum_{k=1}^{K-1} c(a^k_t) \tag{11}$$

where $c(a^k_t)$ is the confidence of one of the $K-1$ additional actions in the distribution at a step.

Thus far, the synthetic data generated will have the ground truth as the action with the highest confidence. We would like to vary this, and simulate perception errors. We define a perception error (PE) as when the action a_t that has the highest confidence in $Distr(a_t)$ does not match the ground-truth action a^*_t. The perception error *rate* (PER) is the percentage of action distributions that have perception errors. We simulate the PE in a particular action distribution by exchanging the highest confident action with another action in that distribution. A specified PER is achieved by simulating PE in a proportion of the action distributions in each plan trace. The action distributions in which we simulate PE are randomly chosen. In this way, data of different distribution types is generated for testing the effectiveness of our model.

[3] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

C EXPERIMENTS SETTINGS

All of our experiments have the following hyperparameters (thus fair to all models): We set the window $\mathcal{W}$ to one, embedding size to 100 and train each model for 60 epochs. We considered other training epochs from 20 to 80, and decide to use 60 in our experiments. The number of missing actions is one in synthetic evaluation and 10% in video based evaluation, the number of recommendations for each missing action is three, and the number of threads when running the experiment is eight. Specifically, in accuracy evaluation on synthetic dataset, we evaluate the data under two categories: 1) Fixed entropy but varying PERs between (25%, 50%, 75%, and 100%); 2) Zero PER but test with both high entropy (all confidence values each time step are uniformly distributed) and low entropy. We set PER to zero in order to evaluate with only the influence of entropy. For low entropy, we set the confidence for a_t^* to 0.9, and the two simulated actions to 0.05. And on real world dataset (consists of whole distribution sequences with varying lengths), we also evaluate the data under two categories: 1) Fixed sequence length (30) but varying number of samples; 2) Fixed number of samples (15) but varying sizes of distributions.

We also look at the effect of the number of sampled paths on the training time, on the synthetic dataset. Please note that in results where the number of samples are varied, it only affects the `RS2Vec` model. The other models are unaffected by resampling and thus only have their average value plotted as a line.

D WORD2VEC AND HIERARCHICAL SOFTMAX

In this section we make a brief review about how the Skip-gram `Word2Vec` works, based on Mikolov et al. (2013), and how it could be used for plan recognition as introduced in Tian, Zhuo, and Kambhampati (2016). To be consistent with the whole paper, we use the term "action" and treat it an equivalence to the term "word" in other papers which introduces `Word2Vec`.

Given a corpora which contains action (word) sequences, a Skip-gram model is trained by maximizing the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-\mathcal{W}\leq j\leq \mathcal{W}, j\neq 0}\log p(a_{t+j}|a_t) \tag{12}$$

where T is the length of a sequence, $\mathcal{W}$ is the context window size, a_t is fed as the model input, a_I, and a_{t+j} is used as the target action (word), a_O. The probability $p(a_O|a_I)$ is computed as:

$$p(a_O|a_I) = \frac{exp({v'}_{a_O}^T v_{a_I})}{\sum_{a=1}^{A} exp({v'}_{a}^T v_{a_I})} \tag{13}$$

which is essentially a softmax function. A denotes a vocabulary of all possible actions. Other symbols follows the definition in Equation 12. We can use this equation that computes $p(a_O|a_I)$ to compute $p(a_{t+j}|a_t)$ in Equation 12.

The $p(a_{t+j}|a_t)$ in the `Word2Vec` with hierarchical softmax is calculated in the following manner. The output layer's weight matrix of the regular `Word2Vec` is replaced by a binary tree whose leaf nodes are words in the trained vocabulary. Every node on the path from the root node until the leaf node has a vector, excluding the leaf node. The input action a_i is converted into an embedding h which is the input into the binary tree component. The probability of this input vector h that could go to a particular leaf node is calculated by following the path from the root node to the target leaf node, using the following formula:

$$p(a_{t+j}|a_t) = \prod_{i=1}^{L(a_{t+j})-1} \Big\{\sigma(\mathbb{I}(n(a_{t+j}, i+1) = child(n(a_{t+j}, i))) \cdot v_{n(a_{t+j},i)} \cdot h)\Big\}, \tag{14}$$

where $\mathbb{I}(x)$ is a function that returns 1 if the next node on the path to the target leaf node is on the left of the current node, and -1 if the next node is to the right. $L(a_{t+j})$ is the length of path from

root to the leaf node a_{t+j}, $v_{n(a_{t+j},i)}$ is the vector of the ith node along the path. h is the embedding obtained by multiplying the embedding matrix and vector of the input action a_t. h is the vector that represents the input action in the embedding space. In order to maximize the probability, the vectors of the intermediary nodes are updated with each training sample which has the target action a_t and an action in its context a_{t+j}.

E Derivation of Gradient Update Equations

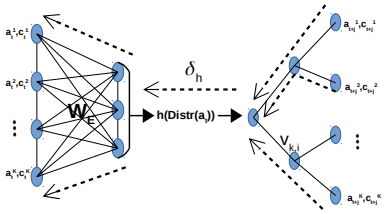

Figure 11: Details of `Distr2Vec` model for Error Propagation.

We back-propagate the error E as shown in Figure 11, where the dashed lines show the path of back-propagation. For this derivation, we shorten $h(Distr(a_t))$ as h. We start by computing the derivative of E with respect to $(v_{n(a^k_{t+j},i)} * h)$ as in Equation 15. We also shorten our notation of $v_{n(a^k_{t+j},i)}$ as $v_{k,i}$, which is the vector of i-th node in the tree path to the leaf node of the k-th action in the vocabulary as illustrated in Figure 11.

$$\frac{\partial E}{\partial v_{k,i}h} = -c^k_{t+j}\frac{\frac{\partial \sigma(\mathbb{I}(.)v_{k,i}h)}{\partial v_{k,i}h}}{\sigma(\mathbb{I}(.)v_{k,i}h)}$$

$$= -c^k_{t+j}\frac{\sigma(\mathbb{I}(.)v_{k,i}h)(1-\sigma(\mathbb{I}(.)v_{k,i}h)\mathbb{I}(.)}{\sigma(\mathbb{I}(.)v_{k,i}h)}$$

$$= c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h)-1)\mathbb{I}(.) = \begin{cases} c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h)-1), (\mathbb{I}(.)=1) \\ c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h), (\mathbb{I}(.)=-1) \end{cases}$$

$$= c^k_{t+j}(\sigma(v_{k,i}h)-t_i) \tag{15}$$

where $t_i = 1$ if $\mathbb{I}(.) = 1$ and $t_i = 0$ if $\mathbb{I}(.) = -1$. Recall that $\mathbb{I}(.)$ is the identity function defined in Equation 14.

Then we calculate the derivative with respect to each $v_{k,i}$ along the path to a specific action, at the time step $t + j$:

$$\frac{\partial E}{\partial v_{k,i}} = \frac{\partial E}{\partial v_{k,i}h}\frac{\partial v_{k,i}h}{\partial v_{k,i}} = c^k_{t+j}(\sigma(v_{k,i}h)-t_i)h \tag{16}$$

With this, we update each $v_{k,i}$ as follows:

$$v_{k,i} = v_{k,i} - \alpha\frac{\partial E}{\partial v_{k,i}} \tag{17}$$

Note that each node's vector $v_{k,i}$ could get updated more than once, as each node could be on the path to more than one action as show in the right side of Figure 11.

Then we compute the back-propagated error δ_h by substituting Equation 15 as follows:

$$\begin{aligned}\delta_h = \frac{\partial E}{\partial h} &= \sum_{k=1}^{K} \sum_{i=1}^{L(a^k_{t+j})-1} \frac{\partial E}{\partial v_{k,i} h} \frac{\partial v_{k,i} h}{\partial h} \\ &= \sum_{k=1}^{K} c^k_{t+j} \sum_{i=1}^{L(a^k_{t+j})-1} \sigma(v_{k,i} h) - t_i) v_{k,i} \end{aligned} \tag{18}$$

We can understand this equation by imagining that there are multiple channels coming back from each leaf node, passing thorough a sequence of child nodes in the hierarchical softmax tree, towards the output of the embedding matrix W_E. So doing the back-propagation means summing errors of these channels together, and that is why $v_{k,i}$ could get updated more than once.

We derive the Equation 18 leveraging Equation 15. However, we could also go directly from the original error function (Equation 8). Thus here we provide another derivation of δ_h which is equally valid:

$$\begin{aligned}\delta_h = \frac{\partial E}{\partial h} &= \frac{\partial [- \sum_{k=1}^{K} c^k_{t+j} \sum_{i=1}^{L(a^k_{t+j})-1} \log \sigma(\mathbb{I}(.) v_{n(a^k_{t+j},i)} \cdot h)]}{\partial h} \\ &= - \sum_{k=1}^{K} c^k_{t+j} [\sum_{i=1}^{L(a^k_{t+j})-1} \frac{\partial \log \sigma(\mathbb{I}(.) v_{n(a^k_{t+j},i)} \cdot h)}{\partial h}] \\ &= - \sum_{k=1}^{K} c^k_{t+j} [\sum_{i=1}^{L(a^k_{t+j})-1} (\sigma(v_{n(a^k_{t+j},i)} h) - t_i) v_{n(a^k_{t+j},i)}] \end{aligned} \tag{19}$$

where $\frac{\partial \log \sigma(\mathbb{I}(.) v_{n(a^k_{t+j},i)} \cdot h)}{\partial h}$ has already been derived as a part of Equation 15.

Finally, we update the weights in the embedding matrix W_E:

$$\frac{\partial E}{\partial W_E} = \frac{\partial E}{\partial h} \frac{\partial h}{\partial W_E} = \delta_h v^t_a \tag{20}$$

where $v^t_a = \langle c^1_t, c^2_t, ..., c^K_t \rangle$ is the confidence values from $Distr(a_t)$, and $\frac{\partial E}{\partial W_E}$ can be used to update the values in W_E as follows:

$$W_E = W_E - \alpha * \frac{\partial E}{\partial W_E} \tag{21}$$