Recognizing Plans by Learning Embeddings from Observed Action Distributions

Handling Uncertainty in Visual Perception for Plan Recognition (Supplemental Material)

Paper ID: 1213

1 Dataset Collection

1.1 Collection of Visual Grounded Plan Corpus

The procedure for making the visual 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 *et al.* [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%.

1.2 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 [1]. 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 1.

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

where a_t^k is the k^{th} additional action in a distribution at step t. We search for a_t^k in the `Word2Vec` model based on the ground truth action a_t^*. s_k is the similarity between a_t^*, and a_t^k, 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 1. 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 $c(a_t^k)$ to

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

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

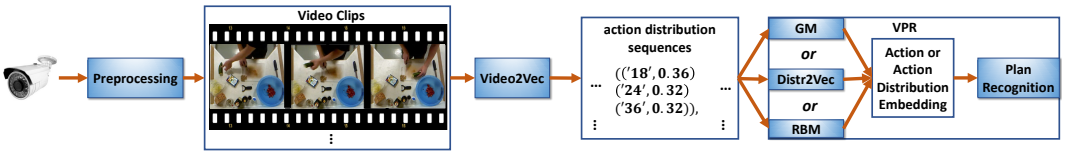

Figure 1: This figure illustrates the framework of VPR, which does plan recognition on uncertain observations. Those uncertain observations comes from a visual recognition model, Video2Vec. We would explain in detail how we train a Video2Vec on real-world videos to do action recognition in Section 1.1. The number '18', '24', and '36' means the 18-th, 24-th, and 36-th action in action vocabulary .

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 2.

$$c(a_t^*) = 1 - \sum_{k=1}^{K-1} c(a_t^k) \tag{2}$$

where $c(a_t^k)$ 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.

2 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 80 epochs. 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 RBM model. The other models are unaffected by resampling and thus only have their average value plotted as a line.

3 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 *et al.* [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:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-\mathcal{W}\leq j\leq \mathcal{W}, j\neq 0}\log p(a_{t+j}|a_t) \tag{3}$$

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:

$$p(a_O|a_I) = \frac{exp({v'}_{a_O}^T v_{a_I})}{\sum_{a=1}^{A} exp({v'}_{a}^T v_{a_I})} \tag{4}$$

which is essentially a softmax function. A denotes a vocabulary of all possible actions. Other symbols follows the definition in Equation 3. We can use this equation that computes $p(a_O|a_I)$ to compute $p(a_{t+j}|a_t)$ in Equation 3.

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:

$$p(a_{t+j}|a_t) = \prod_{i=1}^{L(a_{t+j})-1} \Big\{\sigma(\mathbb{I}(n(a_{t+j}, i+1) = child(n(a_{t+j}, i))) \cdot v_{n(a_{t+j},i)} \cdot h)\Big\}, \tag{5}$$

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}.

4 Derivation of Gradient Update Equations

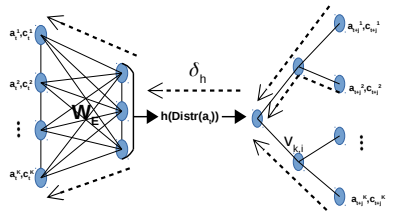

Figure 2: Details of `Distr2Vec` model for Error Propagation.

We back-propagate the error E as shown in Figure 2, 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 $(v_{n(a_{t+j}^k,i)} * h)$ as in Equation 6. 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 2.

$$\begin{aligned}\frac{\partial E}{\partial v_{k,i}h} &= -c^k_{t+j}\frac{\frac{\partial \sigma(\mathbb{I}(.)v_{k,i}h)}{\partial v_{k,i}h}}{\sigma(\mathbb{I}(.)v_{k,i}h)} \\ &= -c^k_{t+j}\frac{\sigma(\mathbb{I}(.)v_{k,i}h)(1-\sigma(\mathbb{I}(.)v_{k,i}h)\mathbb{I}(.)}{\sigma(\mathbb{I}(.)v_{k,i}h)} \\ = c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h)-1)\mathbb{I}(.) &= \begin{cases} c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h)-1), (\mathbb{I}(.)=1) \\ c^k_{t+j}(\sigma(\mathbb{I}(.)v_{k,i}h), (\mathbb{I}(.)=-1) \end{cases} \\ &= c^k_{t+j}(\sigma(v_{k,i}h)-t_i) \end{aligned} \tag{6}$$

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 5.

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

$$\frac{\partial E}{\partial v_{k,i}} = \frac{\partial E}{\partial v_{k,i}h}\frac{\partial v_{k,i}h}{\partial v_{k,i}} = c^k_{t+j}(\sigma(v_{k,i}h)-t_i)h \tag{7}$$

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

$$v_{k,i} = v_{k,i} - \alpha\frac{\partial E}{\partial v_{k,i}} \tag{8}$$

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 2.

Then we compute the back-propagated error δ_h by substituting Equation 6 as follows:

$$\begin{aligned}\delta_h = \frac{\partial E}{\partial h} &= \sum_{k=1}^{K}\sum_{i=1}^{L(a^k_{t+j})-1}\frac{\partial E}{\partial v_{k,i}h}\frac{\partial v_{k,i}h}{\partial h} \\ &= \sum_{k=1}^{K} c^k_{t+j}\sum_{i=1}^{L(a^k_{t+j})-1}\sigma(v_{k,i}h)-t_i)v_{k,i}\end{aligned} \tag{9}$$

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 9 leveraging Equation 6. However, we could also go directly from the original error function (Equation **??**). Thus here we provide another derivation of δ_h which is equally valid:

$$\begin{aligned}\delta_h = \frac{\partial E}{\partial h} &= \frac{\partial[-\sum_{k=1}^{K} c^k_{t+j}\sum_{i=1}^{L(a^k_{t+j})-1}\log\sigma(\mathbb{I}(.)v_{n(a^k_{t+j},i)}\cdot h)]}{\partial h} \\ &= -\sum_{k=1}^{K} c^k_{t+j}[\sum_{i=1}^{L(a^k_{t+j})-1}\frac{\partial\log\sigma(\mathbb{I}(.)v_{n(a^k_{t+j},i)}\cdot h)}{\partial h}] \\ &= -\sum_{k=1}^{K} c^k_{t+j}[\sum_{i=1}^{L(a^k_{t+j})-1}(\sigma(v_{n(a^k_{t+j},i)}h)-t_i)v_{n(a^k_{t+j},i)}]\end{aligned} \tag{10}$$

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 6.

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

$$\frac{\partial E}{\partial W_E} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial W_E} = \delta_h v^t_a \tag{11}$$

where $v_a^t = \langle c_t^1, c_t^2, ..., c_t^K \rangle$ 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:

$$W_E = W_E - \alpha * \frac{\partial E}{\partial W_E} \tag{12}$$

References

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

Sheng-Hung Hu, Yikang Li, and Baoxin Li. Video2vec: Learning semantic spatial-temporal embeddings for video representation. 2016.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

S. Stein and S. J. McKenna. 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, September 2013.

Xin Tian, Hankz Hankui Zhuo, and Subbarao Kambhampati. Discovering underlying plans based on distributed representations of actions. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1135–1143. International Foundation for Autonomous Agents and Multiagent Systems, 2016.