Differentiable Grammars for Videos

AJ Piergiovanni, Anelia Angelova, Michael S. Ryoo
Robotics at Google
{ajpiergi, anelia, mryoo}@google.com

Abstract

This paper proposes a novel algorithm which learns a formal regular grammar from real-world continuous data, such as videos. Learning latent terminals, non-terminals, and production rules directly from continuous data allows the construction of a generative model capturing sequential structures with multiple possibilities. Our model is fully differentiable, and provides easily interpretable results which are important in order to understand the learned structures. It outperforms the state-of-the-art on several challenging datasets and is more accurate for forecasting future activities in videos. We plan to open-source the code.[1]

Learning a formal grammar from continuous, unstructured data is a challenging problem. This is especially challenging when the elements (i.e., terminals) of the grammar to be learned are not symbolic or discrete (Chomsky 1956; 1959), but are higher dimensional vectors, such as representations from real world data sequences such as videos.

Simultaneously, addressing such challenges is necessary for better automated understanding of sequential data. In video understanding, such as activity detection, a convolutional neural network (CNN) (e.g., (Carreira and Zisserman 2017)) generates a representation abstracting local spatio-temporal information at every time step, forming a temporal sequence of representations. Learning a grammar reflecting sequential changes in video representations will enable explicit and high-level modeling of temporal structure and relationships between multiple occurring events in videos.

In this paper, we propose a new approach of modeling a formal grammar for videos in terms of learnable and differentiable neural network functions. The objective is to formulate not only the terminals and non-terminals of our grammar as learnable representations but also the production rules, which are generated here as differentiable functions. We provide the loss function to train our differentiable grammar directly from data, and present methodologies to take advantage of it for recognizing and forecasting sequences[2]. Rather than focusing on non-terminals and production rules to generate or parse symbolic data (e.g., text strings), our approach allows learning of grammar representations directly on top of higher-dimensional data stream (e.g., representation vector sequences). We confirm such capability experimentally by focusing on learning a differentiable regular grammar from continuous representations, which can be applied to any sequential data including outputs of 3-D CNNs.

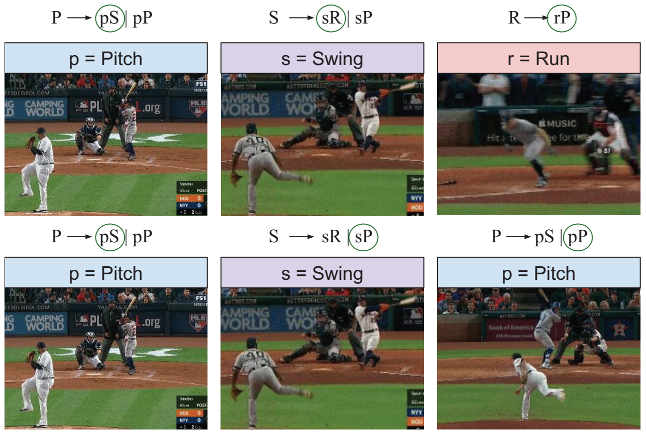

Figure 1: Example regular grammar giving the sequence of possible activities in a baseball video. For example, a swing (s) only occurs after a pitch (p). After a swing (s), both pitch (p) and run (r) are possible continuations of the activity.

Fig. 1 shows an example of consecutive events in video which are learned as grammar rules, e.g. "pitch" is followed by a "swing", which in turn can develop in a "run" event or a another "pitch". This not only allows for better recognition of human activities from videos by enforcing the learned grammar to local-level detections, but also enables forecasting of future representations based on the learned production rules. It also provides semantic interpretability of the video recognition and prediction process.

Our differentiable grammar could be interpreted as a particular form of recurrent neural network (RNN). The main difference to the standard RNNs such as LSTMs and GRUs (Hochreiter and Schmidhuber 1997; Cho et al. 2014) is that

[1]https://sites.google.com/view/differentiable-grammars

[2]Technically the grammar we are learning is a *stochastic* regular grammar, we use regular grammar for simplicity in the text.

our grammar explicitly maintains a set of non-terminal representations, in contrast to having a single hidden representation in standard RNNs, and learns multiple distinct production rules per non-terminal. This not only makes the learned model more semantically interpretable, but also allows learning of temporal structures with multiple sequence possibilities. Our grammar, learned with a randomized production rule selection function, considers multiple transitions between abstract non-terminals when matching it with the input sequences as well as when generating multiple possible future sequences.

In the experiments, we observe much better performance on state-of-the-art on detection tasks on three video datasets, and much better accuracy on video forecasting. We also experimentally compare our grammar with previous related models including LSTMs (Hochreiter and Schmidhuber 1997) and Neural Turing Machines (NTMs) (Graves, Wayne, and Danihelka 2014) in the experiments section.

The primary contributions of our work are:

- We propose a **fully differentiable** model that is able to learn the structure (terminals, non-terminals, and production rules) of a regular grammar. This is done while still maintaining the **interpretability** of the learned model and its representations.
- We show that the model is able to achieve better results on **forecasting** of human activities which are to occur subsequently in videos.
- We confirm that the approach works on sequential real-world datasets, and outperforms the state-of-the-art on **challenging benchmarks**.

The goal of this work is to provide to the research community a neural differentiable grammar-based matching and prediction for video analysis, which is also applicable to other domains. We observe that learning a grammar for continuous video data greatly benefits its automated understanding. The results are interpretable which is very important for real-life decision making scenarios. Furthermore, it can predict future events with higher accuracy, which is crucial for anticipation and reaction to future actions, for example for an autonomous robot which interacts with humans in dynamic environments.

Related work

Chomsky grammars (Chomsky 1956; 1959) are designed to represent functional linguistic relationships. They have found wide applications in defining programming languages, natural language understanding, and understanding of images and videos (Socher et al. 2011). In its original form, a grammar is composed of production rules generating discrete symbols. Machine learning of formal grammar given training data has been traditionally known as grammar induction (Fu 1977). Here, our main motivation is to newly design a differentiable version of a grammar representation whose parameters could be optimized with a standard backpropagation (together with the other components) for its learning.

There are early works exploring extracting grammars/state machines from trained RNNs (Kolen 1994; Bodén and Wiles 2000; Tiňo et al. 1998). Other works have attempted to learn 'neural push down-automata' to learn context-free grammars (Sun et al. 2017) or neural Turing Machines (Graves, Wayne, and Danihelka 2014). However, these works only explored simple toy experiments, and were not tested on real-world data. Our work is also related to the fields of differentiable rule learning (Yang, Yang, and Cohen 2017; Evans and Grefenstette 2018), the literature review of which goes beyond the scope of the paper.

Some works have explored learning more explicit structures by forcing states to be discrete and uses pseudo-gradients to learn grammatical structures (Zeng, Goodman, and Smyth 1994). However, they still rely on a standard RNN to learn model the sequences. It has also been found that LSTMs/RNNs are able to learn grammars (Gers and Schmidhuber ; Giles, Horne, and Lin 1995; Das, Giles, and Sun 1992). Unlike these works, we design a neural network architecture that is able to explicitly model the structure of a grammar, which leads to much easier interpretability.

Other works have explored using neural networks to learn a parser. (Socher et al. 2011) parse scenes by learning to merge representations. (Mayberry and Miikkulainen 1999) learn a shift-reduce neural network parser and (Chen and Manning 2014) learn a dependency parser as a neural network. While these works learn grammar structures, they are generally difficult to interpret.

Within the activity recognition domain, regular and context-free grammars have been tranditionally used to parse and understand videos (Moore and Essa 2002; Pirsiavash and Ramanan 2014; Ivanov and Bobick 2000; Ryoo and Aggarwal 2009; Si et al. 2011). Other works have extended CFGs such as attribute grammars (Joo and Chellappa 2006) or using context-sensitive constraints and interval logic (Brendel, Fern, and Todorovic 2011; Kwak, Han, and Han 2014). We similarly develop our approach for video understanding tasks. However, none of such grammars were differntiable, preventing their end-to-end learning.

Background

A formal grammar G is defined with four elements: $G = (V, \Sigma, P, S)$ where V is a finite set of non-terminals, Σ is a finite set of terminals, P is a finite set of production rules, and S is the starting non-terminal.

In a regular grammar, the production rules P are in the following forms:

$$\begin{aligned} A &\rightarrow aB \\ A &\rightarrow a \\ A &\rightarrow \epsilon \end{aligned} \tag{1}$$

where A and B are non-terminals in V, a is any terminal in Σ, and ϵ denotes an empty string. A regular grammar is a type 3 formal grammar in the Chomsky hierarchy.

In this paper, we follow this traditional regular grammar definition, while extending it by making its terminals, non-terminals, and production rules represented in terms of differentiable neural network functions.

Approach

Formulation

We model our formal grammar in terms of latent representations and differentiable functions mapping to representations. The parameters of our functions define production rules, which are learned together with the terminal and non-terminal representations. For example, in the context of videos terminals can be per-frame activity labels of a video (multi-labels are allowed too), whereas the role of the production grammar rules is to learn plausible (stochastic) transitions between activities in time (Fig. 1). The non-terminals are learned internally within the grammar and do not have specific interpretation.

Each non-terminal in V is a latent representation with fixed dimensionality, whose actual values are learned based on the training data. Each terminal in Σ corresponds to a video representation that could be obtained at every time step, such as a vector with activity class predictions. This has to be learned as well. Our production rules are represented as a pair of two functions:

- f: a function that maps each non-terminal in V (e.g., A) to a subset of production rules (i.e., the rules that 'expand' the current non-terminal) $\{p_i\} \subset P$.
- g: a function that maps each rule p_i to a terminal (e.g., a) and the next non-terminal (e.g., B).

$$\begin{aligned} f &: V \rightarrow \{P\} \\ g &: P \rightarrow (V, \Sigma). \end{aligned} \tag{2}$$

The combination of the two functions effectively captures multiple production rules per non-terminal, such as "$A \rightarrow aB$" and "$A \rightarrow aA$". The starting non-terminal S is learned to be one of the latent representations in V. The functions are learned from data.

These form a straight forward (recursive) generative model, which starts from the starting non-terminal $S = v^0$ and iteratively generates a terminal at every time step. The system internally keeps states of possible rules and non-terminals and learns their relationships with terminals according to data, starting from a feature representation of the video, non-terminal 'states' v are learned, each one of which learns to generate a number of rules. Each rule is then expanded into a set of non-terminal and terminal, where the terminal can now be compared to the actually observed data (terminal value). This process continues recursively. In the appendix, we illustrate this process and how the loss is computed.

Representing our production rules as functions allows us to model the generation of a sequence (i.e., a string) of terminals as the repeated application of such functions. At every time step t, let us denote the first function mapping each non-terminal to a set of production rules as $k = f(v^t; \theta_1)$, and the second function mapping each rule to a non-terminal/terminal pair as $(v^{t+1}, w^t) = g(p_i; \theta_2)$ where $v \in V$, and $w \in \Sigma$. k is a latent vector describing the production rule activations corresponding to v^t.

In its simplest form, we can make our grammar rely only on one production rule by applying the softmax function (σ) to the activation vector k: $p_i = \sigma(k)$. This formulation makes p_i a (soft) one-hot indicator vector selecting the i-th production rule. Our sequence generation then becomes:

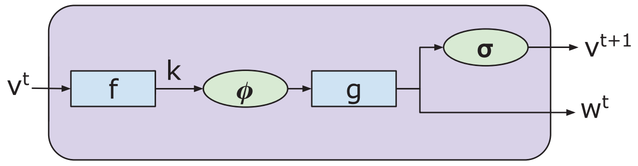

Figure 2: Illustration of the connection between functions in the grammar model. ϕ is the gumbel-softmax function and σ is the softmax function.

$$\left(v^{t+1}, w^t\right) = g\Big(\sigma\big(f(v^t; \theta_1)\big); \theta_2\Big). \tag{3}$$

We represent each $v \in V$ as a N-dimensional soft one-hot vector where N is the number of non-terminals. In the actual implementation, this is constrained by having a softmax function as a part of g_{θ_2} to produce v^{t+1}. Each $w \in \Sigma$ is a T-dimensional representation we learn to generate, where T is the dimensionality of the sequential representation at every time step. This process is shown in Fig. 2.

We further extend Eq. 3 to make the grammar consider multiple production rules in a randomized fashion during its learning and generation. More specifically, we use the Gumbel-Softmax trick (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017) to replace the softmax in Eq. 3. Treating the activation vector k as a distribution over production rules, the Gumbel-Softmax (ϕ) allows sampling of different production rules:

$$\left(v^{t+1}, w^t\right) = g\Big(\phi\big(f(v^t; \theta_1)\big); \theta_2\Big). \tag{4}$$

In our case, this means that we are learning the grammar production rules which could be selected/sampled differently even for the same non-terminal (i.e., v^t) while still maintaining a differentiable process (Fig. 3).

The idea behind our grammar formulation is to allow direct training of the parameters governing generation of the terminals (e.g., video representations in our case), while representing the process in terms of explicit (differentiable) production rules. This is in contrast to traditional work that attempted to extract grammar from already-trained standard RNNs (Gers and Schmidhuber) or more recent neural parsing works using discrete operators (Dyer et al. 2016) and memory-based RNNs (Graves, Wayne, and Danihelka 2014). Our formulation also adds interpretability to our temporal models learned from data streams, as we confirm more in the following subsections.

Detailed implementation of production rule functions: Although any other differentiable functions could be used for modeling our functions f and g, we use matrix operations to implement them. Given a matrix of production rules, W, a $N \times (R \cdot N)$ matrix, where R is the maximum number of production rules per non-terminal, we obtain the activation vector k with size $R \cdot N$ as:

$$k = f(v) = vW \tag{5}$$

We constrain W so that its each column is a vector with only one non-zero element (i.e., each production rule may originate from only one non-terminal). In the actual implementation, W is obtained by modeling it as a $N \times R$ matrix and then inflating it with zeros to have the form of a block diagonal matrix of size $N \times (R \cdot N)$ with the block size $1 \times R$.

Similarly, the function g mapping each production rule to the next non-terminal and corresponding terminal is implemented using a $(R \cdot N) \times N$ matrix H_1, and a $(R \cdot N) \times T$ matrix H_2:

$$(v^{t+1}, w^t) = g(v^t) = (\sigma(FH_1), FH_2) \tag{6}$$

where $F = \phi(f(v^t))$. With this implementation, learning the grammar production rules is done by learning the matrices W, H_1, and H_2 directly. Fig. 4 describes an example.

Learning

We train our grammar model to minimize the following binary cross entropy loss:

$$\mathcal{L} = \sum_{t,c} z_c^t \log(w_c^t) + (1 - z_c^t) \log(1 - w_c^t) \tag{7}$$

where z^t is the ground truth label vector at time t with dimensionality $|c|$ and w^t is the output of the grammar model (terminal). In the case where the grammar is used to predict discrete class labels, z^t becomes a one-hot vector. Training of our functions f and g (or matrices W, H_1, and H_2) can be done with a straight forward backpropagation for the simple production rule case of Eq. 3, as it becomes a deterministic function per non-terminal at each time step. Backpropagating through the entire sequential application of our functions also allow learning of the starting non-terminal representation $S = v^0$.

Learning multiple production rules: In general, our function f maps a non-terminal to a 'set' of production rules where different rules could be equally valid. This means that we are required to train the model by generating many sequences, by taking b rules at each step (b is the branching factor).

We enumerate through multiple productions rules by randomizing the production rule selection by using the Gumbel-Softmax trick (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017) as suggested in the above subsection. This allows for weighted random selection of the rules based on the learned rule probabilities. In order the train our grammar model with the Gumbel-Softmax, we maintain multiple different 'branches' of non-terminal selections and terminal generations, and measure the loss by considering all of them. Algo. 1 and Fig. 3 illustrate the training and branching process. When generating many branches, we compute the loss for each generated sequence, then take the minimum loss over the b branches, effectively choosing the branch that generated the most similar string:

$$\mathcal{L} = \min_b \sum_{t,c} z_{t,c} \log(w_{b,c}^t) + (1 - z_{t,c}) \log(1 - w_{b,c}^t) \tag{8}$$

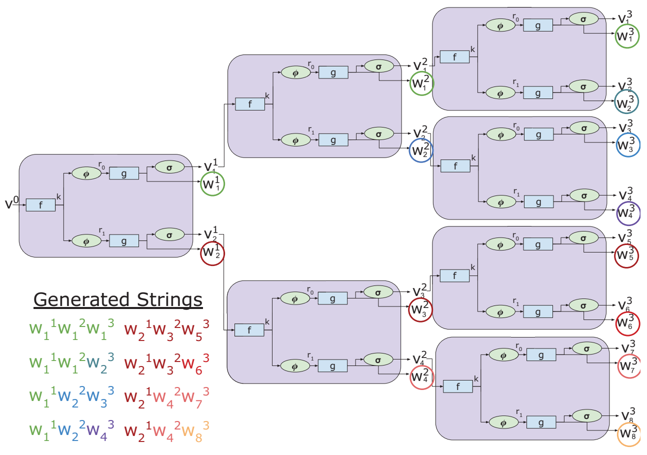

Figure 3: Visualization of training the grammar with branching. The output (r_i) of the Gumbel-Softmax (ϕ) is different for each branch, producing different strings.

Algorithm 1 The training of the grammar, with multiple branches

```
Input: sequence $s$
Set initial nonterminal $v^0$
for $t = 0$ to $T$ do
    for $c = 0$ to current total branches do
        Get rules for current nonterminal: $k = f(v_c^t)$
        for $b = 0$ to Number of branches do
            Randomly select a rule: $p = \phi(k)$
            Get next non-terminal and terminal
            $(v_b^{t+1}, w_b^t) = g_{\theta_2}(p)$
        end for
    end for
end for
$loss = \min_b \mathcal{L}(s, w_b)$, min over all branches
```

where $w_{b,c}^t$ is the output of the grammar model (terminals) at time t for class c and branch b. Branches are pruned to make the process computationally tractable, limiting the total number of branches we maintain.

Interpretability

As our model is constrained to use a finite set of non-terminals, terminals and production rules, it allows for easy interpretability of the learned grammar structure. We can conceptually convert the learned production rule matrices W, H_1, and H_2 into a discrete set of symbolic production rules by associating symbols with the learned terminal (and non-terminal) representations. The matrix W describe the left-hand side non-terminal of the production rule following the regular grammar (e.g., $\mathbf{A} \rightarrow aB$), the matrix H_2 describes the terminal of the production rule (e.g., $A \rightarrow \mathbf{a}B$), and the matrix H_1 corresponds to the right-hand side non-terminal of the rule (e.g., $A \rightarrow a\mathbf{B}$). Element values of the matrix W in particular suggests the probability associated with the production rule (i.e., it governs the probably of the corresponding production rule being randomly selected with Gumbel-Softmax). Fig. 4 shows how we can construct a grammar from the learned matrices.

Figs. 5, 6 illustrate examples of such interpreted grammar, learned from raw datasets. This was done by associating symbols with w^t and v^t. We note that the interpretability (Doshi-Velez 2017) of the model is inherent in our method and comes as an effect of the grammar representation used. More specifically, our method is able to embed grammar-like rules into differentiable learning, while still preserving the interepretability of it.

Application to video datasets

Videos contain complex continuous data from with rich visual time-correlated features. We here describe how to apply the grammar model to videos.

The video is first processed by a backbone model which extracts feature representations of chunks of video sequence (e.g., we use the popular I3D model (Carreira and Zisserman 2017)). The initial non-terminal is learned based on the video representation. We learn a function ψ that maps from the video representation to the initial non-terminal: $S = v_0 = \psi(q)$, where q is the output of the video CNN. We then train the grammar model as above, where the ground truth is the sequence of one-hot vector based on the activity labels in the video.

During inference (which is about predicting frame-level activity labels), we generate a sequence by selecting the rule that best matches the CNN predicted classes. We then multiply the predictions from the grammar with the predictions from the CNN. To predict future actions, we generate a sequence following the most likely production rules.

Experiments

Toy Example

We first confirm that our model is able to learn the rules of a simple, hand-crafted grammar and show how we can easily interpret the learned model. Given the grammar:

$$A \rightarrow aB$$
$$B \rightarrow bC \mid bA$$
$$C \rightarrow cA$$

We train a model with 3 terminal symbols (a, b, and c), 3 non-terminal symbols (A, B and C), and 2 production rules per non-terminal. We can then examine the learned grammar structure, shown in Fig. 4. We observe that the learned starting non-terminal corresponds to 'A', and by following the learned rules, we end up with 'aB'. From non-terminal 'B', the learned rules go to 'bA' or 'bC' with 50% probability. From non-terminal 'C', the learned rules go to 'cA'. This confirms that the model is able to learn grammar rules and can easily be interpreted.

The approach is general and applies also to any streaming data.

Activity Detection Experiments

We further confirm that our method works on real-world, challenging activity detection datasets: MLB-YouTube (Piergiovanni and Ryoo 2018a), Charades (Sigurdsson et al. 2016b), and MultiTHUMOS (Yeung et al. 2015). These datasets are evaluated by per-frame mAP. We also compare on 50 Salads (Stein and McKenna 2013) measuring accuracy. The datasets are described as follows:

MultiTHUMOS: The MultiTHUMOS dataset (Yeung et al. 2015) is a large scale video analysis dataset which has frame-level annotations for activity recognition. It is a challenging dataset and supports dense multi-class annotations (i.e. per frame), which are also used here for both prediction and ground truth. It contains 400 videos or about 30 hours of video and 65 action classes.

Charades: The Charades dataset (Sigurdsson et al. 2016b) is a challenging dataset with unstructured activities in videos. The videos are everyday activities in a home environment. It contains 9858 videos and spans 157 classes.

MLB-YouTube: The MLB-YouTube dataset (Piergiovanni and Ryoo 2018a) is a challenging video activity recognition dataset collected from live TV broadcast baseball games, Fig. 1, . It further offers the challenge of fine-grained activity recognition as all potential activities are encountered in the same context and environment, unlike many other datasets which feature more diverse activities which may also use context for recognition. It has 4290 videos in 42 hours of video. Additionally, baseball games follow a rigid structure, making it ideal to evaluate the learned grammar.

50 Salads: This dataset contains 50 videos of making salads, e.g., cutting vegetables and mixing them.

Implementation Details We implemented our models in PyTorch. The learning rate was set to 0.1, decayed every 50 epochs by 10, and the models were trained for 400 epochs. We pruned the number of branches to 2048 by random selection. The number of grammar parameters vary by dataset driven by the number of classes, MLB has 8 terminals (for 8 classes), 5 rules per non-terminal, 8 non-terminals. Charades - 157 terminals, 10 rules per non-terminal, 1000 non-terminals. The LSTM has 1000 hidden units for all.

Results on MLB-Youtube

Table 1 shows the results of the proposed algorithm on the MLB-Youtube dataset, compared to all state-of-the-art algorithms including RNNs such as LSTMs. We evaluated the methods in two different settings: 1) learning grammar on top of features learned from I3D and 2) on top of a recently proposed super-events method. The result clearly shows that our differentiable grammar learning is able to better capture temporal/sequential information in videos. We also compare to LSTMs and NTMs using both CNN features (e.g., I3D) as input and using the predicted class probabilities as input, as that is more comparable to our grammar model. We find that the use of class probabilities slightly degrades performance for LSTMs and NTMs.

Results on MultiTHUMOS

Table 2 shows results comparing two common methods with and without the proposed grammar. We also test both settings as above and compare to the state-of-the-art. In both settings we can see that use of the learned grammar outperforms previously known methods.

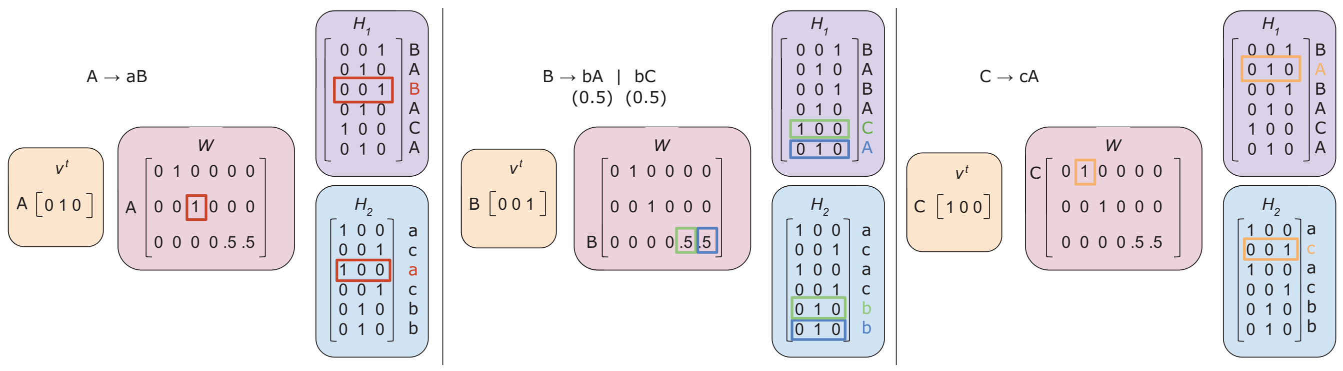

Figure 4: Visualization of the learned toy grammar and how we can construct the grammar from the learned matrices. The non-terminal v^t gives a soft-index into the rule matrix W, which gives probabilities over the rules. The rules give a soft-index into the non-terminal matrix (H_1) and terminal matrix (H_2).

Table 1: Detection results on the MLB-YouTube dataset (mAP).

Model	mAP
Random	13.4
I3D	34.2
I3D + LSTM	39.4
I3D + NTM (Graves et al. 2014)	36.8
I3D class prob + LSTM	37.4
I3D class prob + NTM	36.8
I3D with Grammar (ours)	**43.4**
I3D + super-events (Piergiovanni and Ryoo 2018b)	39.1
I3D + super-events with Grammar (ours)	**44.2**

Table 2: Detection results on the MultiTHUMOS dataset (mAP).

Method	mAP
Two-stream (Yeung et al. 2015)	27.6
Two-stream + LSTM (Yeung et al. 2015)	28.1
Multi-LSTM (Yeung et al. 2015)	29.6
Predictive-corrective (Dave et al. 2017)	29.7
I3D baseline	29.7
I3D + LSTM	29.9
I3D + NTM (Graves et al. 2014)	29.8
I3D class prob + LSTM	29.8
I3D class prob + NTM	29.7
I3D with Grammar (ours)	**32.3**
I3D + super-events (Piergiovanni and Ryoo 2018b)	36.4
I3D + super-events with Grammar (ours)	**37.7**
I3D + TGM (Piergiovanni and Ryoo 2019)	46.4
I3D + TGM with Grammar (ours)	**48.2**

Results on Charades

Table 3 has results comparing the proposed grammar to other prior techniques on the Charades dataset (v1_localize setting). As seen, this dataset is quite challenging since recently its detection accuracy was below 10 percent mAP.

Table 3: Detection results on the Charades dataset (Charades_v1_localize setting).

Method	mAP
Predictive-corrective (Dave et al. 2017)	8.9
Two-stream (Sigurdsson et al. 2016a)	8.94
Two-stream+LSTM (Sigurdsson et al. 2016a)	9.6
R-C3D (Xu, Das, and Saenko 2017)	12.7
Sigurdsson et al. (Sigurdsson et al. 2016a)	12.8
I3D baseline	17.2
I3D + LSTM	18.1
I3D + NTM	17.5
I3D class prob + LSTM	17.6
I3D class prob + NTM	17.4
I3D with Grammar (ours)	**18.5**
I3D + super-events (Piergiovanni and Ryoo 2018b)	19.4
I3D + super-events with Grammar (ours)	**20.3**
I3D + TGM (Piergiovanni and Ryoo 2018b)	22.3
I3D + TGM with Grammar (ours)	**22.9**

Our results here too outperform the state-of-the-art, increasing the accuracy on this dataset. We note that there are consistent improvements in both settings, similar to the results on MultiTHUMOS and MLB-YouTube. In particular, the differentiable grammar learning outperformed previous RNNs including LSTMs and NTMs.

Future Prediction

As our grammar model is generative, we can apply it to predict the future, unseen activities. Future prediction is important, especially for autonomous systems (e.g., robots) as they need to anticipate potential future activities to respond to. Once the grammar is learned, future sequences containing unseen activities can be generated by selecting the most probable production rule at every (future) time step.

For this experiment we consider predicting at short-term horizons (in the next 2 seconds), mid-term horizons (10 seconds), and more longer-term horizons (20 seconds). We compare to baselines such as random guessing, repeatedly

Table 4: Future prediction on the MultiTHUMOS dataset for various time horizons.

Method	2 sec	10 sec	20 sec
Random	2.6	2.6	2.6
Last frame	16.0	12.7	8.7
I3D + LSTM	15.7	6.8	4.1
I3D + Grammar (ours)	**18.6**	**12.8**	**10.5**

Table 5: Future prediction on the Charades dataset for various time horizons.

Method	2 sec	10 sec	20 sec
Random	2.4	2.4	2.4
Last frame	13.8	11.2	**8.6**
I3D + LSTM	12.7	10.8	7.0
I3D + Grammar (ours)	**14.8**	**11.2**	8.5

predicting the last seen frame, and an LSTM approach (with I3D features) which is commonly used for future frame forecasting. We evaluate these methods using per-frame mAP.

Table 4 shows the results for future prediction on the MultiTHUMOS dataset. We confirm the grammar generates more accurate futures. We note that 10-20 seconds in the future is a very challenging setting to try to predict especially in the context of multi-label datasets.

Table 5 shows the results for future prediction for the Charades dataset. Here too, we can see the proposed grammar approach is more accurate at future frame prediction.

Following the setting in (Ke, Fritz, and Schiele 2019), we evaluate the future prediction task on the 50 Salads dataset (Stein and McKenna 2013). The results are shown in Table 6. The results confirm that our approach allows better prediction.

Previous work on video forecasting, e.g., (Yeung et al. 2015) did not address longer horizons. Other work on early detection is also related (Ryoo 2011; Ma, Sigal, and Sclaroff 2016; Wang and Hoai 2018), but they only focused on early detection of an ongoing event rather than forecasting multiple longer-term future events.

Visualization of Learned Grammars

In Fig. 4, we illustrate how we convert from the learned matrices to the grammar and production rules. From the training data, we know the mapping from terminal symbol to label. We can then examine the rule matrix, W and the non-terminals, H_1 to construct the rules.

Fig. 5 visualizes example learned grammar rules for the Charades dataset. We can see learned grammar rules corresponding to natural sequences of events.

We also visualize the learned grammar for the MLB-YouTube dataset, in which, since it is smaller, we can extract all the learned rules. Fig. 6 is the conceptual visualization of the learned regular grammar. Interestingly the typical baseball sequences are learned. More specifically, in Fig. 6, we see that that the learned grammar matches the structure in a baseball game and the probabilities are similar to the observed data, confirming that our model is able to learn the correct rule structure. For example, an activity starts with a pitch which can be followed by a swing, bunt or a hit. After a hit, foul, or strike, another pitch follows. The learned grammar is illustrated with probabilities for each rule in parenthesis. In the appendix, we illustrate the actual learned matrices corresponding to one of the production rules.

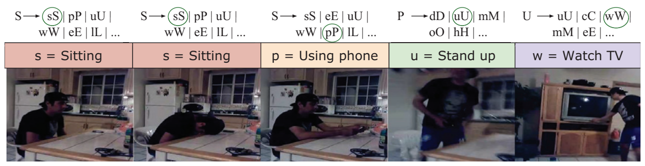
S ⟶ sS | pP | uU | wW | eE | lL | ...
S ⟶ sS | pP | uU | wW | eE | lL | ...
S ⟶ sS | eE | uU | wW | pP | lL | ...
P ⟶ dD | uU | mM | oO | hH | ...
U ⟶ uU | cC | wW | mM | eE | ...
s = Sitting
s = Sitting
p = Using phone
u = Stand up
w = Watch TV

Figure 5: Examples of a sequence and grammar structure learned from the Charades dataset. Other possible activities include e=eating, h=holding cup, m=making sandwich, l=using laptop, etc.

$$\begin{array}{lllllll}
P \longrightarrow & \textit{pitch}\ S & | \ \textit{pitch}\ B & | \ \textit{pitch}\ K & | \ \textit{pitch}\ U & | \ \textit{pitch}\ Y \\
 & (0.5) & (0.25) & (0.21) & (0.03) & (0.01) \\
S \longrightarrow & \textit{swing}\ H & | \ \textit{swing}\ K \\
 & (0.68) & (0.32) \\
U \longrightarrow & \textit{bunt}\ H & | \ \textit{bunt}\ K \\
 & (0.76) & (0.24) \\
H \longrightarrow & \textit{hit}\ P & | \ \textit{foul}\ P \\
 & (0.42) & (0.58) \\
B \longrightarrow & \textit{ball}\ P \\
K \longrightarrow & \textit{strike}\ P \\
Y \longrightarrow & \textit{hit by pitch}\ P
\end{array}$$

Figure 6: The learned grammar from the MLB-Youtube dataset. For non-terminals with multiple rules, the learned probabilities are in parenthesis.

Table 6: Results on 50 Salads without ground-truth observations (setting in Table 2 of (Ke, Fritz, and Schiele 2019)).

Observation	20%				30%			
Prediction	10%	20%	30%	50%	10%	20%	30%	50%
RNN	30.1	25.4	18.7	13.5	30.8	17.2	14.8	9.8
CNN	21.2	19.0	16.0	9.9	29.1	20.1	17.5	10.9
TCA (Ke et al.)	32.5	27.6	21.3	16.0	35.1	27.1	22.1	15.6
Grammar	**39.2**	**32.1**	**24.8**	**19.3**	**38.4**	**29.5**	**25.5**	**18.5**

Conclusion

In conclusion, we presented a differentiable model for learning formal grammars for the purposes of video understanding. The learned structures are interpretable which is important for understanding the behavior of the model and the decisions made. The proposed method outperforms all prior state-of-the-art techniques on several challenging benchmarks. Furthermore, it can predict future events with higher accuracy, which is necessary for anticipation and reaction to future actions. In the future we plan to apply it to even longer horizon data streams. Further, we aim to enable application of our differentiable grammar learning to higher-dimensional representations, learning them jointly with image and video CNNs in an end-to-end fashion.

References

Bodén, M., and Wiles, J. 2000. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*.

Brendel, W.; Fern, A.; and Todorovic, S. 2011. Probabilistic event logic for interval-based event recognition. In *CVPR*.

Carreira, J., and Zisserman, A. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*.

Chen, D., and Manning, C. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, 740–750.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.

Chomsky, N. 1956. Three models for the description of language. In *IRE Transactions on Information Theory (2)*.

Chomsky, N. 1959. On certain formal properties of grammars. In *Information and Control. 2 (2)*.

Das, S.; Giles, C. L.; and Sun, G.-Z. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Conference of Cognitive Science Society*.

Doshi-Velez, Finale; Kim, B. 2017. Towards a rigorous science of interpretable machine learning. In *arXiv:1702.08608*.

Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent neural network grammars. In *NAACL-HLT*.

Evans, R., and Grefenstette, E. 2018. Learning explanatory rules from noisy data. In *JAIR*.

Fu, K. S. 1977. *Syntactic Pattern Recognition, Applications*.

Gers, F. A., and Schmidhuber, J. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*.

Giles, C. L.; Horne, B. G.; and Lin, T. 1995. Learning a class of large finite state machines with a recurrent neural network. *Neural Networks* 8(9):1359–1365.

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9:1735–80.

Ivanov, Y. A., and Bobick, A. F. 2000. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax. In *ICLR*.

Joo, S.-W., and Chellappa, R. 2006. Attribute grammar-based event recognition and anomaly detection. In *CVPR Workshops*.

Ke, Q.; Fritz, M.; and Schiele, B. 2019. Time-conditioned action anticipation in one shot. In *CVPR*.

Kolen, J. F. 1994. Fool's gold: Extracting finite state machines from recurrent network dynamics. In *NeurIPS*.

Kwak, S.; Han, B.; and Han, J. H. 2014. On-line video event detection by constraint flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(6):1174–1186.

Ma, S.; Sigal, L.; and Sclaroff, S. 2016. Learning activity progression in lstms for activity detection and early detection. In *CVPR*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*.

Mayberry, M. R., and Miikkulainen, R. 1999. Sardsrn: A neural network shift-reduce parser. In *Proceedings of the 16th Annual Joint Conference on Artificial Intelligence*.

Moore, D., and Essa, I. 2002. Recognizing multitasked activities from video using stochastic context-free grammar. In *AAAI*.

Piergiovanni, A., and Ryoo, M. S. 2018a. Fine-grained activity recognition in baseball videos. In *CVPR Workshop on Computer Vision in Sports*.

Piergiovanni, A., and Ryoo, M. S. 2018b. Learning latent super-events to detect multiple activities in videos. In *CVPR*.

Piergiovanni, A., and Ryoo, M. S. 2019. Temporal gaussian mixture layer for videos. In *ICML*.

Pirsiavash, H., and Ramanan, D. 2014. Parsing videos of actions with segmental grammars. In *CVPR*.

Ryoo, M. S., and Aggarwal, J. K. 2009. Semantic representation and recognition of continued and recursive human activities. *IJCV* 82(1):1–24.

Ryoo, M. S. 2011. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *ICCV*.

Si, Z.; Pei, M.; Yao, B.; and Zhu, S.-C. 2011. Unsupervised learning of event and-or grammar and semantics from video. In *ICCV*.

Sigurdsson, G. A.; Divvala, S.; Farhadi, A.; and Gupta, A. 2016a. Asynchronous temporal fields for action recognition. *arXiv preprint arXiv:1612.06371*.

Sigurdsson, G. A.; Varol, G.; Wang, X.; Farhadi, A.; Laptev, I.; and Gupta, A. 2016b. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*.

Socher, R.; Lin, C. C.; Manning, C.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *NeurIPS*.

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*, 729–738. ACM.

Sun, G.-Z.; Giles, C. L.; Chen, H.-H.; and Lee, Y.-C. 2017. The neural network pushdown automaton: Model, stack and learning simulations. *arXiv preprint arXiv:1711.05738*.

Tiňo, P.; Horne, B. G.; Giles, C. L.; and Collingwood, P. C. 1998. Finite state machines and recurrent neural networks – automata and dynamical systems approaches. In *Neural networks and pattern recognition*. Elsevier. 171–219.

Wang, B., and Hoai, M. 2018. Back to the beginning: Starting point detection for early recognition of ongoing human actions. *CVIU*.

Xu, H.; Das, A.; and Saenko, K. 2017. R-c3d: Region convolutional 3d network for temporal activity detection. *arXiv preprint arXiv:1703.07814*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*.

Yeung, S.; Russakovsky, O.; Jin, N.; Andriluka, M.; Mori, G.; and Fei-Fei, L. 2015. Every moment counts: Dense detailed labeling of actions in complex videos. *IJCV* 1–15.

Zeng, Z.; Goodman, R. M.; and Smyth, P. 1994. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks* 5(2):320–330.