Reinforcement Learning for Continuous Sign Language Recognition

ST11: Rohan Panda, Sistla Shashank, Abhinav Krishnan

21 April 2021

1 Introduction

The aim of Continuous Sign Language Recognition (CSLR) is to **translate videos of sign language into text sentences**. CSLR is a fairly challenging task, and is currently an actively growing field. Any CSLR model must capture all the fine details in hand movements and accurately translate them into coherent sentences without semantic gaps. This task of automatic translation is pertinent in today's world, considering the communication challenges that exist between hearing-impaired individuals and those with hearing abilities. Hence, the use of machines for this task can be particularly beneficial.

2 Objective

The paper we implemented was "Continuous Sign Language Recognition Via Reinforcement Learning"[1] [7]. It proposes a novel approach to the task of Sign Language Recognition which applies Reinforcement Learning. The CSLR model proposed in this paper consists firstly of a 3D ResNet (a type of 3D Convolutional Neural Network) to extract the visual features from the videos. Convolutional Neural Networks are the industry standard for video recognition tasks.

The second portion of the model is a Transformer for the purpose of sign language recognition. The Transformer is a state of the art model that helps translate the sign videos into text sentences. Transformers have shown tremendous capabilities in machine translation tasks, which are very similar to the CSLR task that we sought to implement.

The Transformer was trained using the **self-critic REINFORCE algorithm**. Using RL for training addresses most drawbacks that were observed while using traditional Supervised Learning techniques. This RL based optimization strategy is hypothesized to lead to major improvements in results.

[1] Link to repo: `https://github.com/PandaBoi/CSLR_with_RL`

3 Methodology

3.1 Feature Extraction(3D - ResNet)

The universal approximation theorem states that given enough memory and computing speed, a feedforward neural network is sufficient to represent any function. However, this will mean that the layer will be very large and will be prone to overfitting. To combat this, networks are made deeper with fewer neurons with each layer. This comes with it's own issue: the vanishing gradient problem.

The concept of Residual Networks tackles this problem by using a technique called **"skip connections"**, represented in Figure 1. They are the main building blocks of ResNets.

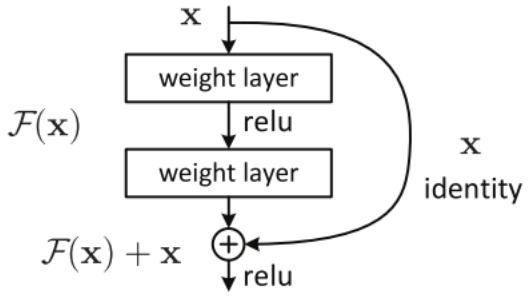
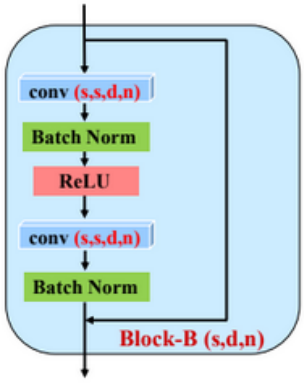


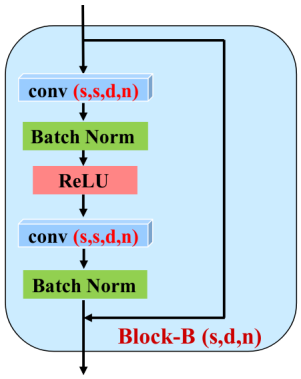
Figure 1: Building block of ResNet: the residual connection.

The outputs of a preceding layer are directly added onto the output of a layer a few layers ahead. This lets regularization skip any layer which reduces the performance of the architecture. This implies that deeper models will not have vanishing gradients, and will be able to outperform shallower models. Variants developed include skipping to before the activations and denser connections between various layers.

3D convolutional neural networks have achieved state-of-the-art performances in action recognition, and is a suitable choice in this use case [2, 4]. The "3D" refers to the use of **3-dimensional convolutional filters**, which allows the Resnet to model both spatial information and temporal information between the frames. The implementation in this paper uses an 18-layer Resnet. The last layer is a pooling layer of output dimension 512 is applied to extract features from the inputted frames. The skips occur before the ReLu function is applied, skipping over two convolutional layers, while the skip itself alternates between: 1) an identity matrix, 2) a convolutional filter with 1x1x1 filter size followed by a batch normalization. The two blocks are visualized in Figure 2.



(a) Block A residual structure



(b) Block B residual structure

Figure 2: The two residual connections used in the ResNet which was part of the complete model. The complete model architecture was found in [3]

.

3.2 Neural Machine Translation(Transformers)

The transformer architecture was used for the task of Neural Machine Translation. Transformers are based on the attention model, first introduced in the revolutionary paper **Attention is All You Need**[5]. Attention is a mechanism used for deep neural networks which allow them to implement the action of selectively concentrating on a few relevant things, while ignoring the rest. It emerged as an improvement over traditional Neural Machine Translation systems.

Transformers use the model of Self-attention, which is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations

The encoder-decoder structure is at the heart of most neural sequence transduction problems today, and the transformer is no different. In a transformer, the encoder maps an input sequence of symbol representations to a sequence of continuous representations. The decoder generates an output sequence of symbols one element at a time. At each step the model is **auto-regressive**, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.

The encoder in our implementation takes an input of a feature vector for the video clips. This is a vector of size 512 outputted by the ResNet. This is passed through the attention and feed-forward loops in the encoder, and fed into the decoder. The decoder takes inputs of learned embeddings of the sentences from the encoder or the beginning token of the next sentence. It outputs the probabilities of the predicted next token.

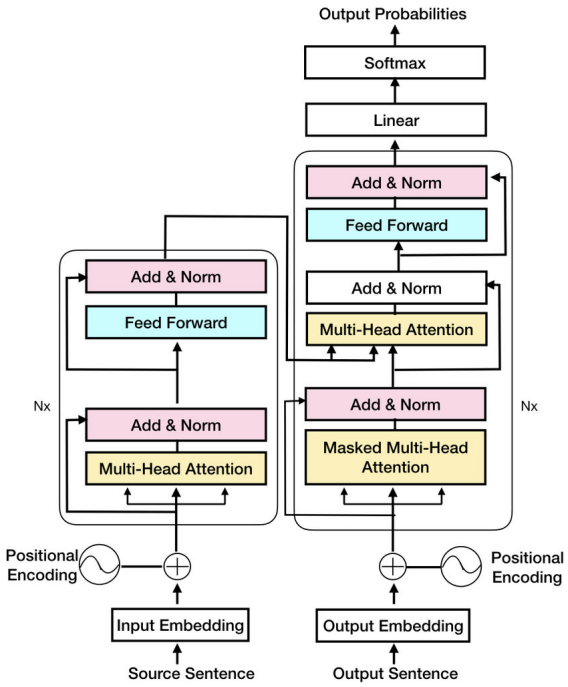


Figure 3: General representation of the Transformer. In this work N = 6 layers were used and feature vectors from the ResNet were inputted instead of source sentence.

3.3 Reinforcement Learning(self-critic REINFORCE)

For problem-cases wherein the state-space and the action-space span a huge region and cover a lot of states and actions, tabular methods and memory-based algorithms fail to provide a feasible method for applying RL. In such situations, model-free RL algorithms are widely utilized. In this paper, a new variant of REINFORCE[6], called as the self-critic REINFORCE, is introduced.

REINFORCE can be considered as the Monte Carlo based algorithm which uses policy gradients to find the optimal probability distribution of actions using a parametric approach. For this use-case a **finite (and discrete) action space, as well as, a stochastic policy in a deterministic environment** is considered. Generally, in supervised learning, models are trained using a cross entropy loss w.r.t. the predictions and labels. However, these might provide divergent performances as compared to non-differential metrics such as **word error rate (WER)**. To be able to incorporate WER as the evaluation metric and incorporate it into the training mechanism of the Transformer, we use $1 - \text{WER}$ [7] as the reward by the environment at the terminal state (denoted by R) and for rest of the states the reward is considered as 0. The Loss function can then be defined as:-

$$L(\theta) = -\mathbb{E}_{w^s \sim p_\theta}[R(w^s)] \tag{1}$$

where $w^s = (w_1^s, \cdots, w_T^s)$ are words sampled from the model at time-steps $t \in [1, T]$. To train the model, the gradient of this loss function is calculated as:

$$\nabla L(\theta) \approx -[R(w^s)\nabla_\theta log p_\theta(w^s)] \tag{2}$$

where the expectation is replaced by the samples of an episode. It is a documented fact that models that use REINFORCE usually have a high variance in the loss values. To tackle this issue, a **baseline factor "b"**, which cannot be dependent on action"w^s", is used in the gradient as,

$$\nabla L(\theta) \approx -(R(w^s) - b)\nabla_\theta log p_\theta(w^s) \tag{3}$$

Since the training happens sample-by-sample, the final gradient for each single step is given by the equation above.

Self-critic REINFORCE: As seen above REINFORCE uses a baseline to decrease the variance during training. This variance (denoted by term "b") can be replaced with parametric values which could be learnt during the training process.

$$\nabla L(\theta) \approx -(R(w^s) - R(\hat{w}))\nabla_\theta log p_\theta(w^s) \tag{4}$$

In self-critic REINFORCE, the baseline term is replaced by the rewards that the agent would receive when it produces the output for the model as an inference. Concretely, self-critic REINFORCE passes the **same training data as a testing data point** and utilizing the rewards received as the baseline. The inference is denoted as $\hat{w} = (\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_T)$ and is derived using:

$$\hat{w}_t = \underset{w_t}{argmax}\ p(w_t). \tag{5}$$

The simple change in the baseline helps the agent in:

- optimizing on sequence-level evaluation metric (WER) instead of cross-entropy loss.
- making the learning of the model context-independent as it gets normalized through the baseline.
- reduces the training parameters and time by making the baseline inferential, thereby improving training speed and reducing complexity.

4 Assumptions and Unknowns

The paper has very sparse details on how they actually implemented their model. This was a significant challenge for us.

The ResNet implemented in the paper was **pre-trained on another dataset**, one that we do not have access to. Neither do we have the computational power to train the same. There was **no mention of the hyperparameters** for the ResNet, which left us in the dark as to how to implement it.

The training process was not explained in the paper. It was unclear whether the training was

done clip-wise, video-wise or batch-wise. Thus, the nature of the input to the Transformer was unclear to us.

The pipeline was vague, and the implementation of the link between the ResNet and the Transformer was not explained. Details about **how the latent features of the ResNet are used in the Transformer are missing**.

The glosses, which are the sentences outputted by the Transformer, needed to be **appended to the hypotheses**. The procedure to do this was also not explained. The conversion of rewards for gloss calculation needed explanation.

To address these concerns, we tried reaching out to the authors of the paper by email, but **we received no response**.

5 Experiments

Following the paper's information the Transformer part of the model was coded and experimented upon using the features extracted from pre-trained ResNet which was found from the git repo of the author of the paper[2]. The following experiments were used to reach the results the paper had showed:

- Since pre-training and fine-tuning of ResNet was not feasible due to **resource restrictions**, the features were used as is and only Transformer was trained along with the glosses available in the dataset.
- As the exact method of training was not clearly explained, we used one 8 frame clip along with its corresponding gloss as input, the outputs were then appended **ignoring the overlapping 4 frames of clips**. This is an assumption taken since it was not discussed in the paper.
- Word embedding could not be used in latent features that were outputted from ResNet as we did not have the required information on how to pass these features into the Transformer. To this end, we made a make-believe embedder using a **single fully connected layer** to mimic the purpose of word embedding. The positional embedding was used as is.
- We strategized to use an ϵ**-greedy method** to enable the model to see more actions before falling prey to the **peakiness effect** [1] which was clearly the reason the model predicted same glosses after a while.

These assumptions were used to build the code of the paper, however there was very little success in replicating the results. One observation that was found is that, the Transformer used to **produce only a single or very few tokens as the output, i.e., it fails at generalizing or exploring the action space properly**. This can be attributed to the

[2]Link to features: `https://github.com/ustc-slr/DilatedSLR`

fact that our method of feeding data into the Transformer was off from what was actually used in paper.

Another interesting observation was that based on the formulation of this training process using 3, the Transformer can stop the training by **"cheating"**. In the particular equation 3, if the model increases the probability of any one gloss to be close to 1, both the sample step and arg-max step would pick the same action. In this way the **loss turns out to be 0** and the training halts completely. This can also be traced back to the method in which the model is trained, we have sent each clip (8 frame sections) of one video from start to end before sending clips from other videos. This might lead the model to converge on such results where it learns to predict only a single action to make the loss 0. We were not able to get across this obstacle and hence were not able to properly train the model.

To try and analyze the reason behind such behaviour we tried to implement the **supervised learning** method of training the Transformer. However, we found that the results were **similar to the RL method**. The model learnt to quickly predict a single type of gloss. Realizing that this must be due to the bias in type of data fed, we tried batch-learning, however owing to computational restrictions we were not able to successfully train the complete model. Nevertheless, sending batches of data reduced the model's tendency to produce the same output for all input. This suggests that the method used in our paper-reproduction differs from the original method, however there is no way to know the actual method employed.

This still does not solve the issue of training the Transformer using RL. It is required that more information on the methodology used in the original paper is available for reproduction of the work.

6 Contributions

The following subsections talk about the contribution of each team member. While the gross distribution was on these lines, the subtler works were distributed equally and each member has worked or touched upon all the aspects in the project. All members have written their share of the report as well.

1. **Rohan Panda (2017A3TS0487H)**:
    - Collected and processed the RWTH-Phoenix Weather dataset.
    - Built the data pipeline, dataset and dataloaders used in the codebase.
    - Implemented self-critic REINFORCE class and integrated it in the main training testing loop.
    - Implemented the SL method of training of the Transformer and ran the mentioned experiments.
2. **Sistla Shashank (2017A1PS0834H)**:
    - Implemented the 3D ResNet that generated feature vectors of the sign clips.

- Coded the environment for the REINFORCE algorithm.
- Built the logger that keeps track of the training of the model.

3. **Abhinav Krishnan(2018AAPS0368H)**:

   - Researched and implemented the Transformer Architecture from scratch for the CSLR task.
   - Worked on the Training and Inference loops that were used by the REINFORCE algorithm to train and test the model.

References

[1] Leshem Choshen et al. "On the weaknesses of reinforcement learning for neural machine translation". In: *arXiv preprint arXiv:1907.01752* (2019).

[2] Shuiwang Ji et al. "3D convolutional neural networks for human action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2012), pp. 221–231.

[3] Junfu Pu, Wengang Zhou, and Houqiang Li. "Dilated Convolutional Network with Iterative Optimization for Continuous Sign Language Recognition." In: *IJCAI*. Vol. 3. 2018, p. 7.

[4] Zhaofan Qiu, Ting Yao, and Tao Mei. "Learning spatio-temporal representation with pseudo-3d residual networks". In: *proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5533–5541.

[5] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[6] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[7] Zhihao Zhang et al. "Continuous Sign Language Recognition via Reinforcement Learning". In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 285–289. DOI: 10.1109/ICIP.2019.8802972.