ASL Translator: Building Intuition for RNN

Xiangyu Liu
Department of Computer Science
Stanford University
jxiangyu@stanford.edu

Abstract

This project simplifies the real world ASL-to-English translation in to a 10-ASL-Signs classification problem. It aims to examine how effective different RNN techniques can be to solve the multi-class classification problem and exhibit the common training and data processing problems when those techniques are applied.

1 Introduction

3.6% of the U.S. population, or about 11 million individuals, consider themselves deaf or have serious difficulty hearing (NDC). And over 6,500 infants are identified as deaf or hard of hearing through newborn screening each year in the United States (HRSA), and more than 90 percent of deaf children are born to hearing parents (NIH).

ASL, American Sign Language, is the sign language most commonly used by the Deaf and Hard of Hearing people in the United States and is also the most effective way for them to communicate. However, learning ASL is hard and time consuming, and only 2.8% of American Adults use ASL (NCBI). Hence, creating an ASL to English translator can drastically reduce the communication barrier with the deaf and hard of hearing people.

Instead of examining the entire ASL language space, this project focuses on a simplified version of the translation problem: 10-class classification. The input to the model is a sequence of frames, each frame contains a list of landmarks for a given ASL sign, representing one of the 10 English words, such as "TV" or "animal". The output from the model is to predict which English word it is.

2 Related Work

Sign Language Translation was first introduced in Camgoz et al. (2018) where an attention-based encoder-decoder neural net joint with CNN based spatial embedding was proposed and the first continuous sign translation dataset, PHOENIX14T, was created. Other subsequent works such as Camgoz et al. (2020), Yin and Read (2020), and Ananthanarayana et al. (2021) further explore using Transformers as the learning technique for sign language translation and have obtained very promising results.

3 Dataset

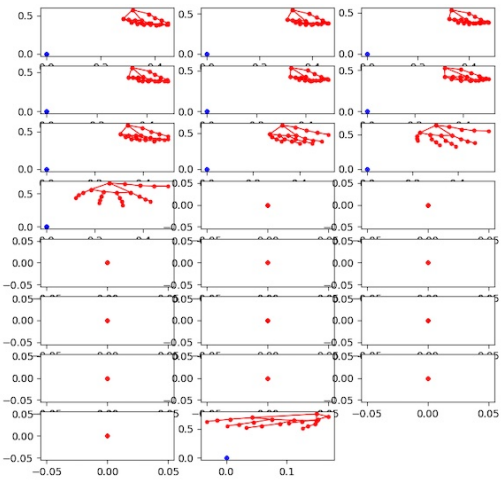
The dataset is directly from Kaggle. Each input data contains a path to a landmark file. A landmark file contains a list of frames from a video where a presenter is signing the given sign. Each frame contains the detected landmarks including hand landmarks (both left and right), face landmarks, and post landmarks.

Each input data also contains the observed y value, a sign, such as “airplane”, “alligator”. In a real world setting, this sign can be any English word or letter. In this project, the problem is reduced to only classifying 10 words ‘[’TV’, ’after’, ’airplane’, ’all’, ’alligator’, ’animal’, ’another’, ’any’, ’apple’, ’arm’]‘.

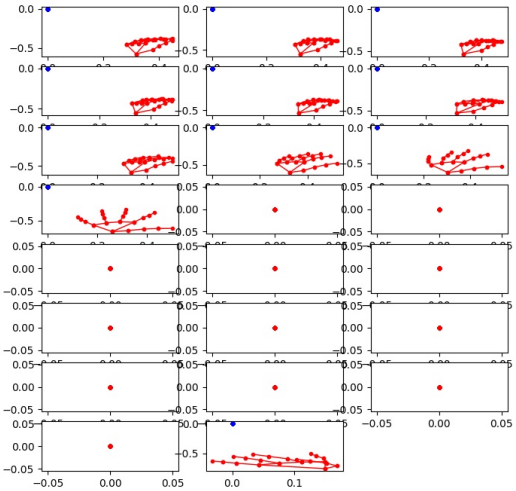
The training data is preprossessed in to training and test dataset using 80-20 split. All training data is further processed in the following way:

Converting NaN: when a hand is not detected, the corresponding landmark values are ‘nan‘, which breaks the training algorithm. Hence, all NaN values are replaced by 0.

Rotation: the landmark values are usually rotated. For example, a “blow” sign looks like this in a video (Language). However, the landmark for a “blow” from the training data is actually rotated, as shown below.



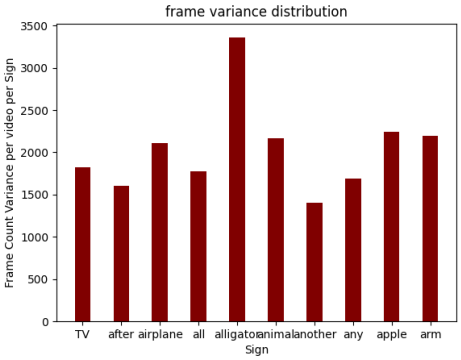
Therefore, a rotation-along-x-axis transformation is applied to all the data (both left and right hands) to obtain the actual hand position.



Noise: It's clear that from the "bird" sign sequence above, there is noise in the training data. The "bird" sign should end with the 4th picture on the first column. However, there is a sequence of empty frames followed by a detected hand (last frame). I did not do anything specific data processing to remove the noise. The model should be able to overcome this type of noise during training.

Other potential issues: There are some other potential issues that can impact the model accuracy and quality:

Different Speed: Just like some people speak fast and some slow, some people sign fast and some slow. This results in different frame counts for different people signing the same word. This results in a high-variance on the frame count for each sign as shown below:



Scaling: The training data contains different presenters signing different words. Different presenters' hands can differ in size. Hence, the landmark data for different people can have different scaling. This is especially significant when training a model to translate signs for both adults and children, because the size of a children's hand can be significantly smaller than an adult's one.

Different Hand: Different people have different dominant hands. Therefore, for the same sign, some data might show left hand while others might show right hands.

4 Methods

4.1 Models

To build up the intuition about how different types of NN behavior, I've created 4 different models to examine how they behave when processing the sequence data. I used the 80-20 split for training and test datasets.

Basic RNN model: the first model is simply a many-to-one RNN model.

$$\sigma = Tanh()$$

$$h^{(t)} = \sigma_h(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + \omega_{hb})$$

$$y^{(t)} = \sigma_y(W_{yh}h^{(t)} + \omega_{yb})$$

RNN + CNN Model: the second model shares the same hidden layer architecture as the first one, however, instead of using σ_y function to compute the final output, it uses a Converlutional Neural Net (CNN) instead. It reuses both σ and $h^{(t)}$ from the first model, but replace σ_y with $CNN(h^{(t)})$, where the $CNN()$ function is simply a sequence of $Conv1D()$, $ReLU()$, $MaxPool1D()$, and $Dropout()$.

Nested CNN + RNN Model: the third model aims to mimic some physical properties of the ASL.

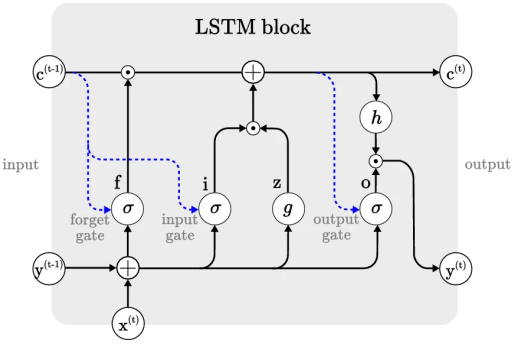
Intuitively, ASL consists the finger movements of each hand and the related movements between both hands. Hence, the third model contains 2 sub models: 1 capturing the features of 1 single hand, another capturing the features of the related movements between both hands.

Sub model 1: As mentioned above, the first model tries to capture the features of 1 hand. Therefore, a CNN is used to extract the features from a single frame. Its output is then used as input to a RNN model in order to process sequence data. In summary, the sub model 1 is represented by the following:

$$\sigma = ReLU()$$
$$f(x^{(t)}) = Dropout(MaxPool1d(\sigma(Conv1d(x^{(t)}))))$$
$$h^{(t)} = \sigma_h(W_{hx}f(x^{(t)}) + W_{hh}h^{(t-1)} + \omega_{hb})$$
$$y^{(t)} = \sigma_y(W_{yh}h^{(t)} + \omega_{yb})$$

Sub Model 2: The second sub model internally instantiates the first sub model twice, 1 model per hand. Then it uses the first model outputs as inputs and feeds them into a CNN to compute the final model output.

LSTM Model: this model leverages a Long Short-Term Memory (LSTM) model internally. A LSTM Model (Hochreiter and Schmidhuber, 1997) is designed to overcome the vanishing gradient problems which typically arises when learning a long input sequence. A typical LSTM (Van Houdt, 2020) is shown below:



Input Gate:

$$i^{(t)} = \sigma_i(W_{ix}x^{(t)} + W_{iy}y^{(t-1)} + w_{bi})$$

Forget Gate:

$$f^{(t)} = \sigma_f(W_{fx}x^{(t)} + W_{fy}y^{(t-1)} + w_{bf})$$

Output/Exposure Gate:

$$o^{(t)} = \sigma_o(W_{ox}x^{(t)} + W_{oy}y^{(t-1)} + w_{bo})$$

New Memory Cell:

$$g^{(t)} = \sigma_g(W_{gx}x^{(t)} + W_{gy}y^{(t-1)} + w_{bg})$$

Final Memory Cell:

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ g^{(t)}$$

Final Hidden Layer Output:

$$y^{(t)} = O^{(t)} \circ \sigma_y(c^{(t)})$$

Final Output:

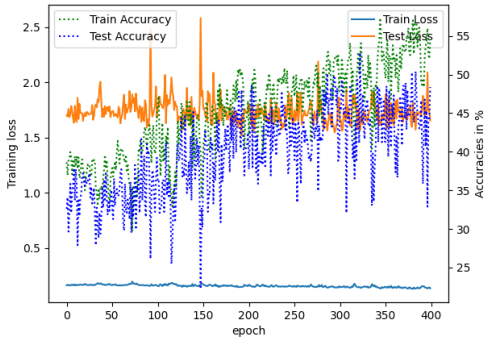
$$z^{(t)} = \sigma_z(W_{zy}y^{(t)} + w_{bz})$$

5 Experiment and Result Discussion

5.1 Vanishing Gradients

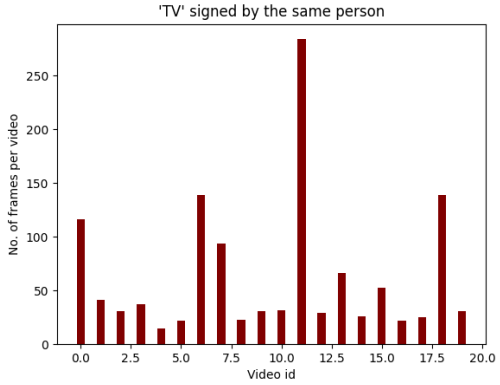
Conceptually, the **Nested CNN+RNN Model** should produce a much better prediction accuracy on both training and test datasets. However, in reality it didn't. It suffered significantly from the Vanishing Gradients problem. Stacking CNN and RNN together exacerbates the problem.

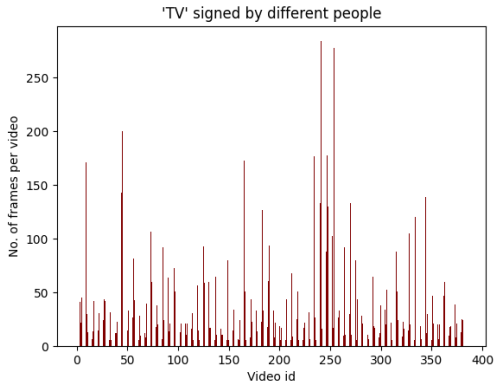
After training with 400 epochs, the train loss has stayed flat, which indicates that the model is not learning from the training. When examining the actual gradients for the weights, they indeed have become 0.



5.2 Batching and Padding

Typically, sequence data contains variable length. This is true especially for ASL. Different signs involves different hand movements. Different people might move their hands in different speed, resulting in different frame numbers. Even for the same person, signing the same sign multiple times can still results in multiple different frames. The following graphs shows the frame count distribution for the same sign 'TV' signed by different people and by the same person multiple times.



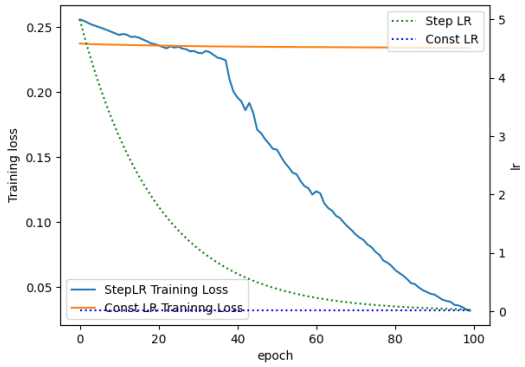


Because of the high variance in the frame count, in order to batch process the inputs, the input with low frame counts are **padded** with zeros in the end to match the highest frame count input in the batch. A smaller batch size can avoid large padding on the batch and thus reduce the average epoch processing time, as shown in the following table.

Batch of 10	Batch of 100
82s	216s

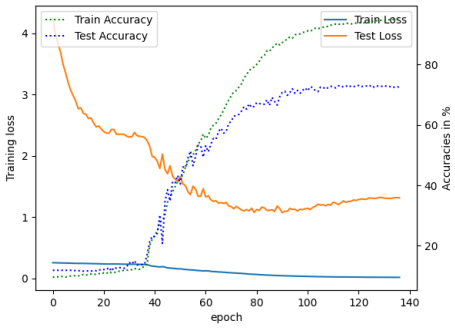
5.3 Learning Rate

Two different LR are experimented with: constant LR with 0.01 and a StepLR with 0.95 gamma decay. As shown below, the constant LR requires a lot more epochs to converge, while the StepLR is able to converge quickly with few epochs.

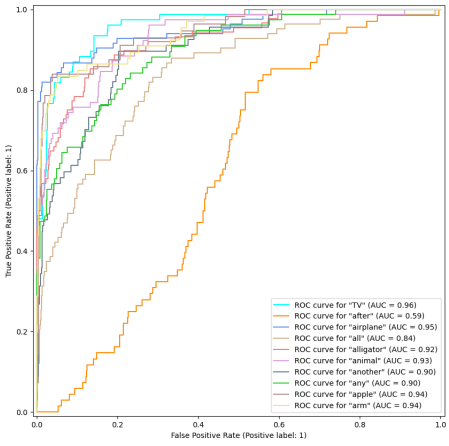


5.4 LSTM Model Results

I'll only focus on the LSTM Model, because it is the only model that was able to converge after training. To avoid model overfitting, I've enabled a Dropout layer in the LSTM. With StepLR, the LSTM model has converged to 90% accuracy on the training set, 0.028 as the train loss, and 73% accuracy with 1.08 as loss on the test set after 100 epochs without any significant overfitting issue.

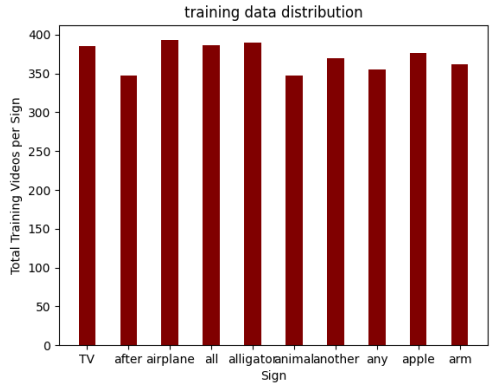


Here is the ROC plot for the 10-class classification LSTM model.

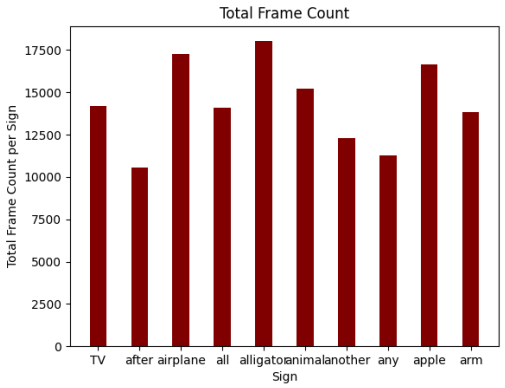


It's quite clear that most of the classes are doing quite well, except the "after" class. I believe it is caused by the training data imbalance.

The "after" class has the fewest training data (videos) when compared to the other classes:



In addition, it also has the least total frame count (sum of all frame counts from all videos):



6 Conclusion and Future Work

When the input is a long sequence, a vanilla RNN will not be able to learn because of the vanishing gradient problem. Therefore, LSTM architecture is crucial to overcome this problem. Especially in this ASL translation problem, the frame count can vary between 50 to 250 frames per sequence, all NNs with vanilla RNNs fail to learn.

Further, even though LSTM did really well for a 10-class classification, its training speed is still very slow, due to the internal for loop. Consider the full training dataset with 94477 input samples over 250 classes, instead of the trimmed version with 3713 input samples over 10 classes. Training 1 epoch takes significantly longer (with the batch size = 10) for the full training dataset, as shown in the table below.

Epoch of 94477 samples	Epoch of 3713 samples
4410s	82s

Therefore, because of its large training latency, training with LSTM becomes impractical for large sequence data. Hence, I would like to further explore the Transformer architecture with multi-attentions to solve such problem.

In addition, I would also like to try some additional preprocessing work, such as upsampling, are necessary to create a balanced training set to reduce the errors caused by the imbalanced data. I would also like to experiment with integrating a CNN into LSTM to leverage its ability of recognizing patterns and see whether it can outperform the vanilla LSTM network. Eventually, I would like to use this model in the real world setting to further examine whether there are additional data biases and how to resolve them.

References

Tejaswini Ananthanarayana, Priyanshu Srivastava, Akash Chintha, Akhil Santha, Brian Landy, Joseph Panaro, Andre Webster, Nikunj Kotecha, Shagan Sah, Thomastine Sarchet, et al. 2021. Deep learning methods for sign language translation. *ACM Transactions on Accessible Computing (TACCESS)*, 14(4):1–30.

Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. 2018. Neural sign language translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7784–7793.

Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. 2020. Sign language transformers: Joint end-to-end sign language recognition and translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

HRSA. Deafness and hearing loss.

Kaggle. Google - isolated sign language recognition.

Baby Sign Language. Blow.

NCBI. How many people use sign language?

NDC. How many deaf people live in the united states?

NIH. Quick statistics about hearing, balance, dizziness.

Mosquera C. Nápoles Van Houdt, G. 2020. A review on the long short-term memory model.

Kayo Yin and Jesse Read. 2020. Better sign language translation with stmc-transformer.