Recurrent Neural Network based Sign Language Recognition using 3D Skeleton Data

Anonymous 3DV submission

Paper ID ****

Abstract

Video based sign language recognition is a mature problem in the computer vision community. With the recent upsurge of depth sensor, $3D$ video data – RGB with depth – has become a popular choice for video analysis because of its extra depth modality. Another type of $3D$ data – $3D$ skeleton data – has shown extensive usage in human activity understanding and become part of state-of-art methods in this domain. Despite having similarity with the human activity recognition, use of $3D$ skeleton data in sign language recognition is scarce. One of the few reasons behind this is unavailability of a benchmark public dataset. In this paper, we introduce a $3D$ skeleton dataset for American sign language (ASL) recognition. The dataset has RGB, depth, skeleton modalities. We also implemented different recurrent neural network (RNN) based architectures and compared with results from dynamic time warping (DTW) based baseline. We evaluated our models in cross subject criteria. In addition to that, we proposed a calibration mechanism for RNN models. Calibration mechanism determines what fraction of train data we need for a test subject to boost models' performance. Experimental results show that, RNN based methods acquire close to 100% test accuracy in most of the cases. The introduced dataset will initiate research in the direction of deep learning sequence modeling for sign language recognition.

1. Introduction

According to National Institute on Deafness and Other Communication Disorders, one in 1000 infants is born totally deaf, while an additional one to six per thousand are born with hearing loss of different levels [10]. Sign language is used by Deaf or Hard-of-Hearing (DHH) people to communicate among themselves. In addition to hearing difficulties, DHH people can have a hardness in understanding and speaking natural languages spoken by normal hearing people [15]. Because of that, they can not use spoken or written form of natural language to communicate with others. Sign language is the natural language to DHH people. The primary use of sign language is to communicate with other DHH people. It also allows them to develop intellectual abilities and learn social traits which are fundamental attributes any normal human being should posses as a part of society.

American sign language (ASL) is the 3^{rd} mostly used language among monolinguals in USA and is used by around half a million of people [?]. Sign language can also be used by the DHH people to communicate with the normal hearing people given that both parties know it. But not many normal hearing people know sign language. That's why daily life communication with people around becomes challenging for them. For example, requesting someone around to open the door, turning on the thermostat or asking someone about weather condition etc. On the other hand, personal digital assistants (PDA) such as Siri, Cortana, Google Home etc. are becoming popular in making life easier for people. These PDAs can be used to make a better quality life for DHH people. To achieve this goal, we must need an ASL recognizer embedded with PDAs.

The purpose of this work is to build an ASL recognition system. Most of the current system dealing with ASL recognition use RGB video data. ASL sign is performed by moving the hands combined with facial expressions and postures of the body. While video data is good for having a total view of whole body movement, we have noticed that, the motion of specific positions of different locations in the hands and head areas are important for ASL recognition. In other word, ASL sign can be treated as a sequence of motion of some specific body parts. Using, video data it is not practical to single out each different body location and motion sequences associated with them from sequences of raw RGB value. Microsoft Kinect is a $3D$ camera sensor which can use the $3D$ information of a person to provide specific $3D$ coordinate of a body location which is called skeletal data [26]. To the best of our knowledge, there is no publicly available skeletal dataset in literature. We used the Kinect sensor to collect data for ASL recognition problem.

With skeleton data, ASL can be seen as a sequence of $3D$ coordinates or a $3D$ time series [3]. Accurate recognition of such data depends on how well we can model inherent sequential pattern in data. Recurrent neural network (RNN) has shown very good performance in type kind of sequential modeling [11]. In this work, we are going to implement two different architectures which are variants of RNN with skeletal ASL data. Success of a practical machine learning system depends on it's performance after it is being deployed. In other words, the system must do well with new subjects other than those were used in training the system. To work in this direction, we proposed a calibration mechanism with our implemented system which essentially finds what fraction of data from a new subject is needed to give a performance boost to our system. Overall we have the following contributions in this work.

- We introduce a new dataset for ASL and make it public [1]
- Implementation of different RNN variants to solve ASL recognition problem with skeletal data.
- Lastly, we propose a calibration mechanism which will tell us what percentage of data from a test subject is needed to give a performance boost to the system.

2. Related work

Most of the sign language recognition systems in the literature use RGB video data as input. These methods use Hidden Markov Models (HMM) to model the sequential dependency for the recognition system. For example Zafrulla *et al*. used worn colored gloves in the hands during data collection and developed an HMM based framework for ASL phrase verification [24]. They also used hand crafted features from Kinect skeleton data and accelerometer worn in hand [23]. Primarily they used directional unit vectors from one joint to another and angle between different combinations of three joints. Having all features crafted, they used an HMM based framework to build the system. Huang *et al*. showed effectiveness of using Convolutional neural network (CNN) with RGB video data for sign language recognition [5]. They used 3D CNN [6] to extract spatio-temporal features from the video and their method does not depend on hand crafted feature for ASL recognition. Similar type of architecture was implemented by Pigou *et al.* for Italian gestures [17]. Sun *et al*. adopted a different way with RGB video data [20]. They hypothesized that, not all RGB frames in a video are equally important and assigned a binary latent variable to each frame in training videos for indicating representativeness of that frame and developed a latent support vector machine model for ASL recognition. Zaki *et al*. proposed two new features with existing hand crafted feature and developed the system using HMM based approach [25]. Cooper *et al*. used appearance based features and divided the whole system into sub units of RGB and tracking data [2]. They also used HMM based model to solve the ASL recognition problem.

To our knowledge, no prior work has used data of tracking body locations alone for ASL recognition problem. However, in a closely similar task of human action recognition, a significant amount of work has been done using tracking body joint information. Sharoudy *et al*. developed the largest dataset in human activity recognition in [18]. They also proposed an extension of long short term memory(LSTM) model which leverages group motion of several body joints to recognize human activity from joint tracking (skeletal) data. A different adaptation of the LSTM model was proposed by Jiu *et al*. [13] where spatial interaction among joints was considered in addition to the temporal dynamics to achieve higher recognition performance. Veeriah *et al*. proposed a LSTM network to capture the salient motion pattern of body joints [21]. This method takes into account the derivative of motion states associated with different body joints. Du *et al*. treated the whole body as a hierarchical configuration of different body parts and proposed a hierarchical RNN to recognize human activities [3]. Some attention based model were also proposed for human activity analysis [14, 19]. There are some methods which converted skeleton sequences of body joints or RGB videos into an image representation and then applied state-of-art image recognition models to achieve good performance [7, 12]. Motivated by successful use of skeleton data in activity recognition, our approach is to perform ASL recognition using skeletal data collected from Kinect sensor.

3. Modeling sign language

Most of the current sign language recognition system use RGB video data as input. Using RGB data there is no way to track a specific body location, such as hand tip, wrist, neck etc. But, to correctly recognize gestures, motion trajectory of each body part could be useful. With the advent of Microsoft Kinect sensor, it is possible to track several specific body locations. Specifically, Kinect version 2 can track 25 body joints. Kinect does this by exploiting depth information in video and using a machine learning model embedded inside it. This process is called skeletal tracking. Figure 1 shows how Kinect sees a person as a configuration of skeletal joints [2]. Since no skeletal dataset was publicly available for ASL, we built our own dataset to initiate research in this direction and made the dataset publicly available.

[1]Email author for code and dataset

[2]https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

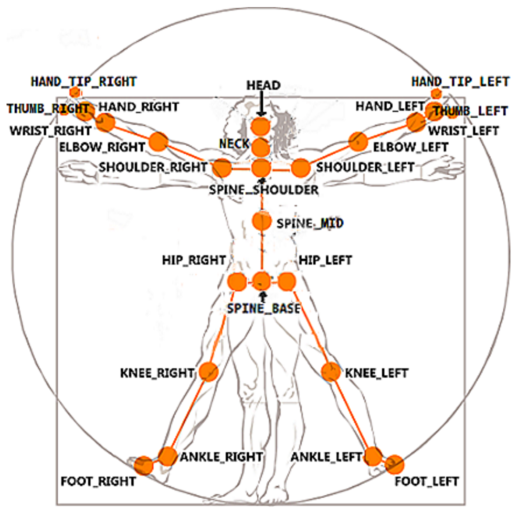


Figure 1: Kinect skeletal body configuration

3.1. Data collection

We collected 480 ASL samples for 4 signers and 6 daily activities related signs. Since our main goal of this work is to help deaf and hearing impaired people, we chose six activities that are considered important for them in daily life. Those are listed in Table 1. For each person, we collected 20 samples for each sign. In this way, we ended up with total $20 \times 6 \times 4$ data samples. We used Kinect version 2 sensor for data collection. For each performer we put the sensor in different distances from the person to make the dataset more challenging and realistic. Figure 2 shows an example of data. It shows six frames from the gesture *air conditioning* at the top and skeletal joints drawn to them at the bottom.

Wake Up	Turn on Thermostat
Interpreter	Air Condition
Open Door	Close Door

Table 1: Selected Commands for data collection and experiment

3.2. Preprocessing

Kinect skeleton sequences are captured in camera coordinate system *i.e.*, camera position is the origin. To make sequences invariant to different camera positions and body-shapes we convert sequences into body coordinate system. We pick a joint location and make it origin, then we subtract (x, y, z) locations of all other joints from it. Finally, different collected sample has different frame length. The video file with 32 frame rate has negligible difference between two consecutive frames. Hence, we sampled a fix number of frames from each sample. We used this fixed number T to be 20.

4. Sequential deep learning for ASL

In this section, we are going to describe several sequential deep learning models and show how we can use those machine learning models for ASL recognition. Like most of the machine learning models, artificial neural networks (ANN) learn a mapping function from an input domain to an output. Usually, inputs to an ANN is a 1D or multidimensional vector and output is a 1D vector. Training procedure starts with initial values for network parameters. The parameters are updated using training data with back propagation algorithm. Speaking of ANN, we usually imagine a feed forward neural network where it has an input layer, an output layer and one or more hidden layers. Parameters in different layers are different from each other. This type of network has shown tremendous performance accuracy in capturing non linear distinction among different classes in a dataset. Specifically, Convolutional neural network (CNN) which is one variant of feed forward neural network has shown human level accuracy for the image classification tasks [9]. Images can be thought of as 2D matrices. Patterns a machine learning model needs to learn from an image extends through this height and width dimensions. Unlike image recognition, some practical problems have also dependency over time *i.e.*, information at certain point depends on information from a previous point or a future point or both. Although feed forward neural networks

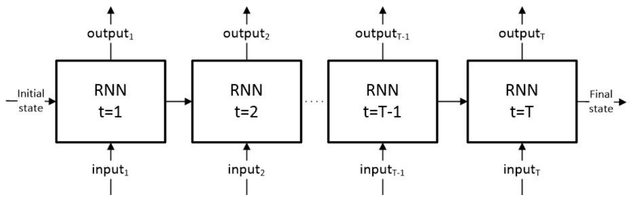


Figure 2: An RNN networks unrolled over T time steps. At each time step the network has one raw input and one processed input from previous time step. Combining these two it produces embedding for current time step

have shown tremendous accuracy in finding spatial pattern in data, it can not efficiently capture such kind of temporal dependencies. One of main reasons behind this weakness is that it sees the whole input data at once and then it keeps transforming the data through its hidden layers until reaches the final layer. In other words it has to capture the patterns as a whole from the input data. To solve sequential modeling of data, Recurrent neural network (RNN) was introduced [11]. It is one of the variations of ANN which is

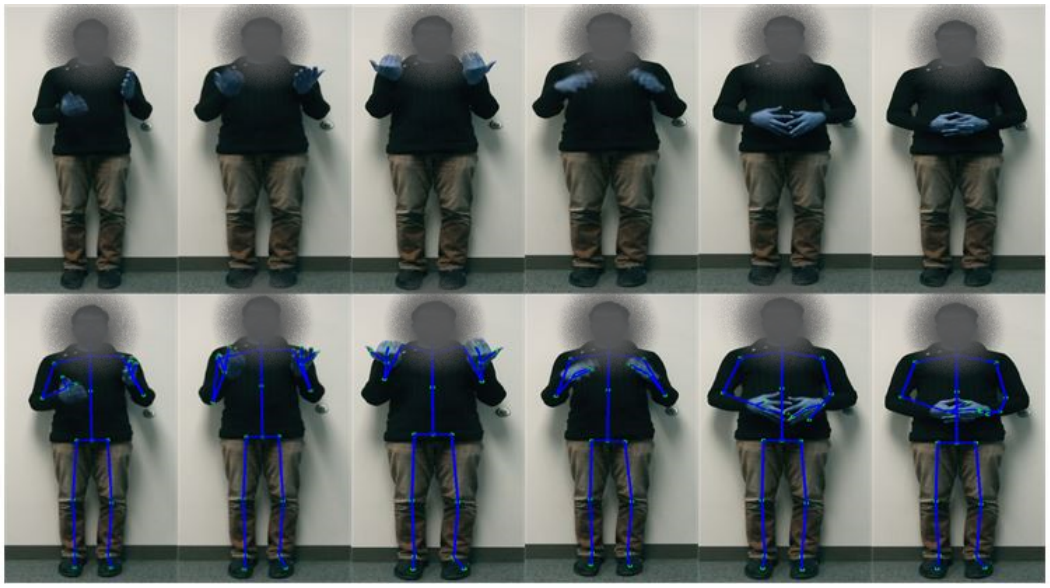


Figure 3: Example of a sign (Air Condition) performed by a person. Top row shows six frames from RGB videos increasing in time from left to right. Bottom row shows exactly same frames but with skeleton data visualization on it. Face of subject intentionally made unrecognizable

suitable for capturing temporal dynamics of data. Figure 3 shows a high level view of such network. Eq 1 shows the equations of an usual RNN. Here x_t is the current input and h_{t-1} is the previous RNN state. U and V are the network parameters associated with x_t and h_{t-1} respectively. ϕ is a nonlinear function which we can choose from several alternatives. Some common choices are $sigmoid$, $ReLu$, $tanh$ etc.

$$\begin{aligned} g(\mathbf{x_t}, \mathbf{h_{t-1}}) &= U\mathbf{x_t} + V\mathbf{h_{t-1}} \\ f &= \phi(g) \end{aligned} \tag{1}$$

Two of the most important differences between RNN and feed forward neural networks are time ordered input and network parameter sharing. We can see from Figure 3 that input is fed to the network in different time steps, not at once like feed forward case. However the parameters the network learns throughout different time steps are same. At any time step the network process current input along with whatever previously seen and processed data. This helps to capture the temporal dynamics in data.

But the basic RNN has problem dealing with long term dependencies in data [16]. This happens because the basic RNN has no control over what it should learn and what should forget. It just keeps updating its memory as it goes which is not desirable in most practical cases. Sometimes a memory from the long past could be useful for current prediction than a memory from the recent past. Another problem which makes training an RNN problematic is known as vanishing gradient problem [16]. Since network parameters of an RNN are shared over time, at any time step the error derivative not only depends on the current input of network but also on the previous state of the network. So there is a multiplicative term of error derivatives back to time step 0 to calculate error derivative of current time. This is a modification of back propagation algorithm and known as back propagation through time (BPTT) [22]. If individual gradients are close to zero this multiplicative term would become very close to zero and eventually gradients disappear. The opposite can also happen, which is called exploding gradient problem, however this is more obvious and easy to detect. The vanishing gradient problem is not easy to detect and serves as a silent killer to the network. Several approaches have been adopted to deal with vanishing gradient problem. Some of them are to careful initialization of network parameters, early stopping etc [16]. But most effective solution was to modify the RNN architecture so that it has its own memory and using that memory it can control what to remember and what to forget. This architecture is called long short term memory (LSTM) network [4]. Another similar architecture called gated recurrent unit (GRU) has been also proposed and has shown robustness against vanishing/exploding gradient problem [1].

4.1. Long short term memory

As we see from Eq 1, at each time step, traditional RNN can be seen as a nonlinear function of the current input and the previous step. While vanilla RNN is a direct transformation from the previous state and the current input, LSTM takes a different approach. It maintains an internal memory and it has mechanism to update and use that memory. The mechanism consists of four separate neural networks also called gates. Figure 4a shows a cell of an LSTM network. Forget, input, update and output gate are four different gates represented by four circles and symbolized as f_t, i_t, $\tilde{C}_t$ and o_t respectively. $\oplus$ and $\otimes$ represents element wise addition and multiplication respectively and $*$ represents matrix multiplication. Two vertical bars inside the rounded rectangle means concatenation operation on its inputs.

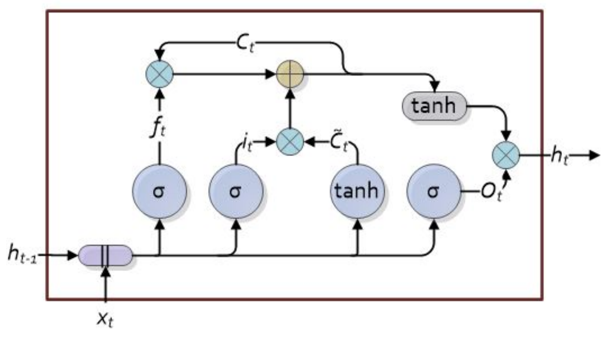
$$\begin{aligned}\mathbf{h_t} &= \mathbf{o_t} \otimes tanh(\mathbf{C_t})\\ \mathbf{o_t} &= \sigma(W_o * concat(\mathbf{h_{t-1}}, \mathbf{x_t}))\\ \mathbf{C_t} &= (\mathbf{f_t} \otimes \mathbf{C_{t-1}}) \oplus (\mathbf{i_t} \otimes \tilde{\mathbf{C}}_\mathbf{t})\\ \mathbf{i_t} &= \sigma(W_i * concat(\mathbf{h_{t-1}}, \mathbf{x_t}))\\ \tilde{\mathbf{C}}_\mathbf{t} &= tanh(W_{\tilde{C}} * concat(\mathbf{h_{t-1}}, \mathbf{x_t}))\end{aligned} \tag{2}$$

Figure 4a shows input at the current time step x_t and the previous state h_{t-1} enter into the cell and get concatenated. Then the forget gate processes it to remove unnecessary information and outputs f_t which gets multiplied with the previously stored memory C_t and produces a refined memory for the current time. Meanwhile, the input gate and the update gate process the concatenated input and convert it into a candidate memory for the current time step. The refined memory from the previous step and proposed candidate memory of the current step get added to produce the final memory for the current step. This addition could render the output out of scale. Because of that, a squashing function is followed which is a hyperbolic tan in our case. Its job is to scale the elements of the output vector into a fix range. Finally o_t, the output from output gate gets multiplied with output from the squashing function and produces the output for the current time step. Eq 2 shows the equations of LSTM where $concat(\mathbf{x}, \mathbf{y})$ means concatenation of input vectors $\mathbf{x}$ and $\mathbf{y}$.

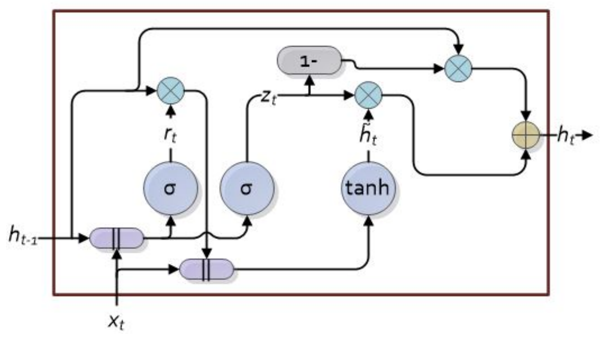
4.2. Gated recurrent unit

A gated recurrent unit (GRU) is another variation of RNN for modeling temporal dynamics of data. It works in the same way as LSTM, but has a simpler construction. Like LSTM it has the mechanism to control flow of information over time, but it lacks the separate memory cell. Figure 4b shows an example of GRU cell. Symbols represent similar things as described in subsection 4.1. To better understand GRU cell, first we have to look at the output of it. The output is formed by a weighted linear sum of two components $\tilde{h}_t$ and h_{t-1} as seen from the Eq. 3.

$$\begin{aligned}\mathbf{h_t} &= (1 - \mathbf{z_t})\mathbf{h_{t-1}} \oplus \mathbf{z_t}\tilde{\mathbf{h}}_\mathbf{t}\\ \mathbf{z_t} &= \sigma(W_z * concat(\mathbf{h_{t-1}}, \mathbf{x_t}))\\ \tilde{\mathbf{h}}_\mathbf{t} &= tanh(W_{\tilde{h}} * concat(\mathbf{r_t} \otimes \mathbf{h_{t-1}}, \mathbf{x_t}))\\ \mathbf{r_t} &= \sigma(W_r * concat(\mathbf{h_{t-1}}, \mathbf{x_t}))\end{aligned} \tag{3}$$



(a)



(b)

Figure 4: a) an LSTM, b) a GRU cell

The term $\tilde{h}_t$ represents newly proposed candidate embedding and h_{t-1} is simply the previous state. Candidate embedding $\tilde{h}_t$ is a function of the current input and purified previous state. This purification is achieved by the reset gate r_t whose job is to decide what portion of previous state should contribute to candidate embedding $\tilde{h}_t$. Reset gate r_t itself is a separate neural network and its output close to 0 means network takes a small portion of previous state to form a current candidate state embedding. The update gate z_t decides what fractions of the candidate state and the previous state comprise the new state. It does so by taking a weighted some of the candidate state and the previous state. The weights are z_t and $1 - z_t$. This whole new state is outputted where in case of LSTM, the output of the new state is controlled by an output gate as shown in Eq. 2.

4.3. Our approach

ASL signs are usually performed using two hands, head movements and in some cases facial expressions. Motion dynamics of two hands are primarily responsible for creating different cues for various signs. Each joint in two hands and face area follows a specific sequential pattern for a particular gesture sign. For example, from Figure 2 we see the person is performing gesture for *air condition*. We can see that, for this sign first, one has to lift and show palm area of the left hand and then he has to lift both hands and make a back and forth motion with all fingers. This is basically a sequence of joints in both hands. Given the efficacy of variants of RNN in encoding sequential pattern in data, we want to see how those models perform on sequence data like ASL sign. To achieve this goal, we implemented LSTM and GRU model for ASL recognition problem. We notice that, the joints located above the waist are mostly useful in case of sign language. Hence, we take only 12 joints into consideration in our experiment. These joints come from both hands, head, neck and spine area. Figure 5 shows our implementation. For each time step, first, we concatenate $3D$ coordinates of 12 joints and then feed to our RNN network. Finally, we take the final state of the network and send it to a softmax layer and get prediction probabilities over our classes.

5. Experiments

In this section, we are going to describe all of our experimental designs. First we are going to explain how we evaluated model performances. We are also going to describe a calibration mechanism we proposed to find a good seed percentage of train data for test person. Then we are going to present implementation details and results we got from experiments.

5.1. Evaluation criteria

Our experiment on the dataset is two fold. First, we did experimentation with two deep learning sequence models, namely GRU and LSTM. Also, we implemented a baseline method based on dynamic time warping (DTW). We proposed a calibration mechanism which essentially infers what fraction of data from an unseen test subject is needed to boost performance of our models. This calibration experiment is done in cross subject manner *e.g.*, we train our models on some subjects and test on a different subject. This makes a lot of sense in real life situation where a trained model has to perform well on data it has not seen yet.

Table 2 represents some terminologies helpful to understand calibration mechanism described next.

We have data on four different subjects. First, we choose a test subject and take out 50% of data from this subject as test data say D_{test}. We use other 50% of data as seed data say, D_{seed} to train model along with the data from other three subjects say, D_{train}. We first train the model only with D_{train} and test on D_{test}. Then we train our model with D_{train} and 10% from D_{seed} and again test on D_{test}. Then again we train the model with D_{train} and this time 20% of D_{seed} and test on D_{test}. As we increase the seed data in our training data, we should expect better test accuracy. We keep doing this until we use up to full D_{seed}

Terms	Description
D_{train}	Training data, whole data from the subjects other than the test subject
D_{test}	Test data, 50% data from test subject used for testing models
D_{seed}	Seed data, 50% data from test subject used for boosting up model's performance on D_{test}

Table 2: Different terminologies regarding data split for one test subject

for training with D_{train} and record different testing results on D_{test}. We repeat the whole procedure by taking each subject as a test subject one at a time. We want to see here what fraction of seed data is required during training to get a good performance from our model. Figure 7 shows data splitting strategy for better understanding.

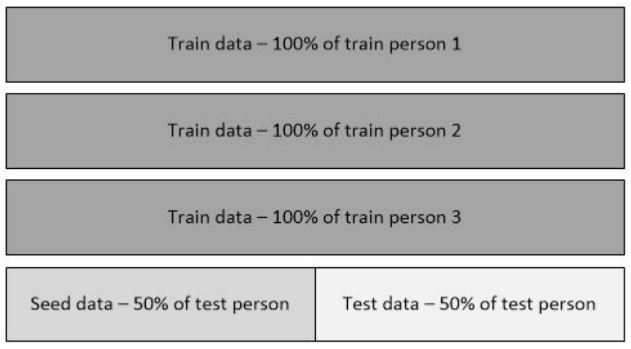

Train data – 100% of train person 1
Train data – 100% of train person 2
Train data – 100% of train person 3
Seed data – 50% of test person
Test data – 50% of test person

Figure 7: Train and test data split for a test person

5.2. Results and discussions

We got an almost similar type of results from both of LSTM and GRU implementation. However, result for DTW implementation is little different. We found that 30% to 50% data from the test subject as seed is good enough to give a performance boost. For DTW, accuracy is relatively lower with 0% seed. But, with small amount of seed (10% − 20%), it reaches highest accuracy for itself. This behavior is desirable because, DTW compares distances of one sample of test set with every other sample in training set and with little introduction of data from same class and subject (seed) would always give minimum distance between test sample and seed. Table 3 shows the results for each combination of a test person, a % of seed data used in training for that person and a network type.

We observe from the result that the highest accuracy is obtained using 30% or more seed data. This suggests that

	% seed	0%	10%	20%	30%	40%	50%
P_1	DTW	0.66	0.89	0.95	0.96	0.96	0.96
	LSTM	0.86	**1.00**	**1.00**	**1.00**	0.99	**1.00**
	GRU	0.78	0.98	0.95	**1.00**	**0.97**	**0.98**
P_2	DTW	0.85	0.89	0.90	0.91	0.94	0.95
	LSTM	0.90	0.93	0.96	0.93	**0.98**	0.97
	GRU	0.93	0.93	0.93	0.93	**0.98**	**0.98**
P_3	DTW	0.50	0.54	0.82	0.94	0.96	**0.96**
	LSTM	0.72	0.65	0.72	0.82	0.82	**0.86**
	GRU	0.62	0.72	0.75	0.86	0.86	**0.90**
P_4	DTW	0.73	0.91	0.91	0.91	0.91	0.91
	LSTM	0.89	0.93	0.93	0.94	0.93	**0.96**
	GRU	0.91	0.94	0.94	**0.96**	0.95	0.95
$\bar{P}$	DTW	0.69	0.81	0.90	0.93	0.94	**0.96**
	LSTM	0.84	0.88	0.90	0.92	0.93	**0.95**
	GRU	0.81	0.89	0.93	**0.94**	0.94	0.95

Table 3: Test accuracies for each combination of test person, method type and % of data from test subject used in training. P_i denotes the i^{th} person and $\bar{P}$ denotes the average accuracy of all subjects

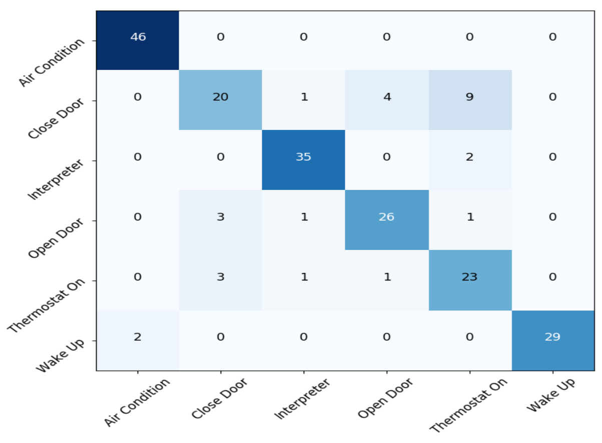
Model type	Test accuracy
LSTM	**0.94**
GRU	0.91

Table 4: Results from 70%–30% train-test split

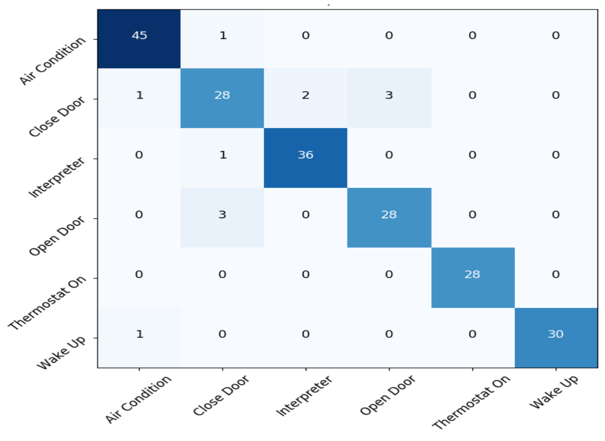
even if we have a small amount of seed data for a test persons, our system can boost up to a higher accuracy. In a practical scenario, system will be able to recognize new subjects correctly within a short period of subject's interaction with the system. Although DTW achieve comparable results in some cases, it is not practical for large operational system because it takes too much time to compute distances with all other samples.

Figure 8a shows confusion matrix when no seed data is used in training for LSTM model. We can see that sign "Close Door" confuses with signs "Thermostat On" and "Open Door". However, in Figure 8b we observe that, with only increase of seed data to 30% in training confusion reduces significantly. We can see from skeleton visualization in Figure 6 that, "Close Door", "Thermostat On" and "Open Door" are somewhat similar in terms of joint movements. That's why the confusion occurs. But, with small amount of seed data, models learn to capture minute details and gain higher accuracy.

We also tested our model in conventional train test split manner. In that case, we took 30% of whole data as test data. Table 4 shows accuracy for both type of RNN models



(a)



(b)

Figure 8: Confusion matrix for 0% (a) and 30% (b) seed data for LSTM model

after performing k-fold cross validation. We did not perform DTW in this experiment since it takes too much time and also depends on distance computation. Our goal of this work is to initiate sequence modeling using deep learning architectures for ASL recognition.

5.3. Implementation and parameter selection

We used same value for hyper-parameters such as learning rate, state size of RNNs, batch size etc. for both compared models. We did a grid search for finding best hyper-parameters. Our experiments found the best learning rate as $5e^{-4}$ and the state size for RNNs as 36. For each joint state size is 3 which gives us the whole state size of 36 for 12 joints. We used Adam Optimizer for optimizing our networks [8]. We used this optimizer because it has shown superior performance in training different types of network. It also adapts learning rate and decay rate as training continues. We used mini-batch size of 1 which means the network updates it's parameter after seeing one training example. We keep training our network until a good accuracy is achieved. However, to battle over-fitting of the network, we followed early stopping strategy. If there is no increase in the testing accuracy for 40 epochs, we decided to stop training. We implemented our models using TensorFlow deep learning library version 1.7 [3]. These experiments were run on ARGO, a research computing cluster provided by the Office of Research Computing at George Mason University, VA [4].

6. Conclusion

A multi-modal dataset for ASL recognition is introduced and made public. Dataset consists of total 480 important signs related to daily activities of deaf or hard-of-hearing people. A cross subject evaluation technique is shown and experimental result shows high testing accuracy. Our proposed calibration mechanism shows what percentage of training data is required from a subject to boost performance. Although, the margin of the dataset is not large, our result suggests a promising direction towards sign language recognition research. Implemented methods show effectiveness of sequential models (RNNs) in practical problem solving. Our future plan is to build a large scale benchmark dataset for ASL recognition by augmenting the current dataset and develop machine learning models that can recognize sentence level ASL.

References

[1] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. 4

[2] H. Cooper, E.-J. Ong, N. Pugeault, and R. Bowden. *Sign Language Recognition Using Sub-units*, pages 89–118. Springer International Publishing, Cham, 2017. 2

[3] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, June 2015. 1, 2

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. 4

[5] J. Huang, W. Zhou, H. Li, and W. Li. Sign language recognition using 3d convolutional neural networks. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, June 2015. 2

[6] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):221–231, Jan. 2013. 2

[7] Q. Ke, M. Bennamoun, S. An, F. A. Sohel, and F. Boussaïd. A new representation of skeleton sequences for 3d action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4570–4579, 2017. 2

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 8

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 3

[10] P. Kushalnagar, G. Mathur, C. J. Moreland, et al. Infants and children with hearing loss need early language access. *The Journal of clinical ethics*, pages 1–1, 2010. 1

[11] Z. C. Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. 1, 4

[12] J. Liu, N. Akhtar, and A. Mian. Skepxels: Spatio-temporal image representation of human skeleton joints for action recognition. *CoRR*, abs/1711.05941, 2017. 2

[13] J. Liu, A. Shahroudy, D. Xu, A. K. Chichung, and G. Wang. Skeleton-based action recognition using spatio-temporal lstm network with trust gates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2017. 2

[14] J. Liu, G. Wang, L. Y. Duan, K. Abdiyeva, and A. C. Kot. Skeleton-based human action recognition with global context-aware attention lstm networks. *IEEE Transactions on Image Processing*, 27(4):1586–1599, April 2018. 2

[15] S. L. Mattys, M. H. Davis, A. R. Bradlow, and S. K. Scott. Speech recognition in adverse conditions: A review. *Language and Cognitive Processes*, 27(7-8):953–978, 2012. 1

[16] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. 4

[17] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen. Sign language recognition using convolutional neural networks. In L. Agapito, M. M. Bronstein, and C. Rother, editors, *Computer Vision - ECCV 2014 Workshops*, pages 572–578, Cham, 2015. Springer International Publishing. 2

[18] A. Shahroudy, J. Liu, T. T. Ng, and G. Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1010–1019, June 2016. 2

[19] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *AAAI Conference on Artificial Intelligence*, pages 4263–4270, 2017. 2

[20] C. Sun, T. Zhang, and C. Xu. Latent support vector machine modeling for sign language recognition with kinect. *ACM Trans. Intell. Syst. Technol.*, 6(2):20:1–20:20, Mar. 2015. 2

[21] V. Veeriah, N. Zhuang, and G. J. Qi. Differential recurrent neural networks for action recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4041–4049, Dec 2015. 2

[22] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct 1990. 4

[23] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti. American sign language recognition with the kinect. In *Proceedings of the 13th International Conference*

[3] https://www.tensorflow.org/
[4] URL:http://orc.gmu.edu

on Multimodal Interfaces, ICMI '11, pages 279–286, New York, NY, USA, 2011. ACM. 2

[24] Z. Zafrulla, H. Brashear, P. Yin, P. Presti, T. Starner, and H. Hamilton. American sign language phrase verification in an educational game for deaf children, 08 2010. 2

[25] M. M. Zaki and S. I. Shaheen. Sign language recognition using a combination of new vision based features. *Pattern Recognition Letters*, 32(4):572 – 577, 2011. 2

[26] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, Apr. 2012. 1

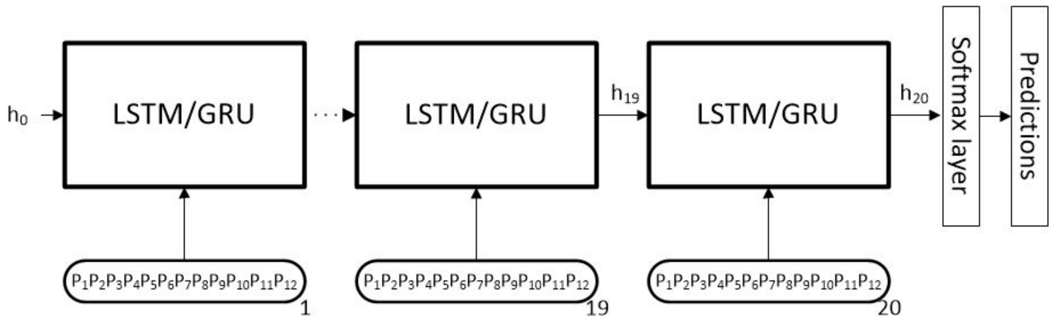


Figure 5: Implementation of RNN models to recognize ASL from skeletal data. Input at each time step is a concatenation of $3D$ coordinates of 12 joints denoted by rounded corner input boxes above. P_i is the (x, y, z) of i^{th} joint. Subscript of each input vector represents time step. We used 20 time steps for each sample data in our experiments as shown above

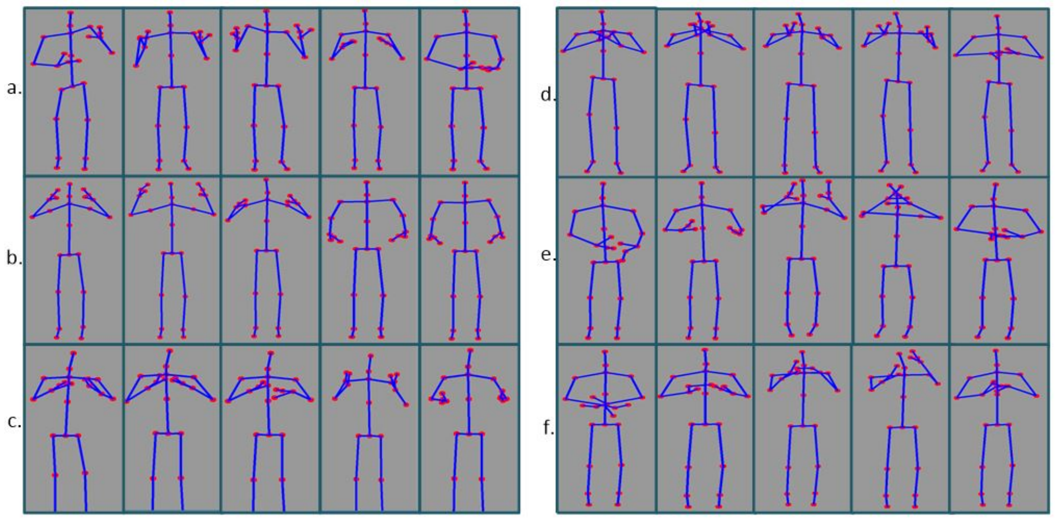


Figure 6: Skeleton visualization of classes in the dataset – a. Air condition, b. Wake up, c. Interpreter, d. Door open, e. Door close, f. Thermostat on