Exploring Deep Recurrent and Spiking Neural Networks in Hand Gesture Recognition

Alwin Lor
lor00148@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Myoelectric Prosthetics are advanced prosthetics that use the electrical signals given off by a user's residual limb to predict what gesture a user intends and then perform that gesture. This paper discusses two methods of gesture recognition that were experimented with and the background knowledge needed to understand them. Deep Recurrent Neural Networks follow a more traditional design and use sequential data to predict hand gestures. Spiking Neural Networks are an uncommon approach to gesture recognition and use sequences of spike-encoded data. Both methods were found to perform well in the metrics chosen for each of them. Both methods are promising ventures into improving hand gesture recognition.

1 Introduction

Modern science and engineering have brought about the creation of more advanced prosthetics. Myoelectric prosthetics is one of these areas of advanced prosthetics. Myoelectric prosthetics use the electrical signals in the residual limb to trigger movement corresponding to what the user intends [10]. This allows users of myoelectric prosthetics to have more advanced and life-like control. The methods used for gesture recognition in commercial prosthetics are proprietary, and information regarding them is not publicly available. This paper will explore two methods studied in academic research: Deep Recurrent Neural Networks (DRNNs) from Aliman et al. [2] and Spiking Neural Networks (SNNs) from Garg et al. [5]. DRNNs are a more traditional way of thinking about neural networks. SNNs offer a more unique and uncommon approach.

In section 2, we present the background information needed to understand Deep Recurrent Neural Networks. Then, we discuss in section 3 Spiking Neural Networks. In section 4, we will focus on how the data is collected and on the results of the two methods. Finally, in section 5, we will present our final thoughts on the two methods.

2 Background

2.1 Neural Networks

Neural Networks (NNs) are machine learning models that are inspired by the way that human brains process information, weigh options, and make decisions [13]. Neural networks are made up of at least three layers of nodes. These layers consist of the input layer, the hidden layer(s), and the output layer. A *feature* is an aspect of the data. The input of an NN is the feature(s) given to the input layer to be used. Information from the input layer is passed to the hidden layer(s). The output of a neural network is the result given in the output layer once the input has been run through the network. Nodes have either one or no activation function. They also have weights on their edges, leading to other nodes. Initially these weights are randomly assigned on all edges. The output of a node is multiplied by the weight before being passed onto the next node.

Before NNs can be used for predictions based on new data, they must be trained on large amounts of already existing data. NNs learn patterns within the data given. NN training can be supervised or unsupervised. In the case of this paper, supervised training will be the main focus. Supervised training is when a NN is given a dataset where all outcomes are known. This is known as a labeled dataset. *Error* comes from the difference in the outcomes from the labeled data set and the network's predicted outcomes [13]. Using backpropagation, the model works backward from the output layer to the input layer to calculate the error of each node and gradually adjusts the weights in the model until the output matches the labeled outcome for all or the majority of the dataset. For example, we input the gesture data of the hand sign "scissors", and the network's output is "rock". In backpropagation, the difference between "rock", the network's output, and "scissors", the true output, is calculated. This is known as the *error*. The error is used to adjust weights backward to get the network's output closer to the true output of the data. It's important to keep in mind that neural networks encode outcomes numerically. Backpropagation is adjusting weights to get incorrect outcomes across all of the data to shift closer to the correct number based on labeled data while keeping correct answers from shifting closer to the incorrect number.

When an algorithm, such as a NN, is trained to separate labeled input data into different classes, it is known as a *classifier*. An example of a classifier NN is an NN that separates and labels images as either a cat or not a cat (binary classifier) or as a cat, a dog, or a bird (multi-class classifier). Once trained, a classifier can use new unlabeled input data

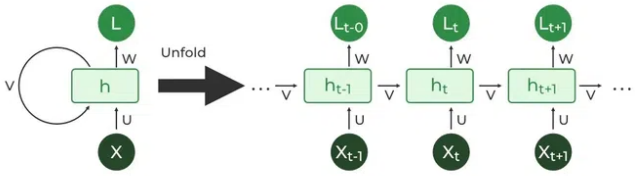


Figure 1. Folded and Unfolded Recurrent Neural Network Node [6]

from the same classes (for example, a picture of a cat the NN hasn't seen) and make a prediction of what class it belongs to. [3]

2.2 Recurrent Neural Networks

RNNs are a variation of neural networks that work with sequential data [14]. A sequence of data can be words in a sentence or hourly temperature readings. A traditional NN takes individual data points, whereas RNNs take into account the order of the sequence when making predictions. For example, an RNN may predict what the next word in a sentence is or what class sequential data belongs to. RNNs accomplish these predictions by having "memory". Each node in the hidden layer can loop back to itself while taking in a new input. The node's input is now the previous layer's output and its own output from its previous calculation. This allows an RNN to take into account previous elements in its sequential data while working with a new input.

For RNNs, the weight parameters are the same each time data is repeated through a node. As seen in Figure 1. *u*, *w*, and *v* are all weights on edges and stay the same as the RNN loops through the input represented as *x*. Once the hidden layer, represented by *h*, has processed the data, the output of *h* is given to the output layer, represented as *L*. The image on the right side of Figure 1 visualizes the "unfolding" of the RNN, representing each time step as a separate path through the RNN. RNNs employ backpropagation through time (BPTT) for training. The difference between BPTT and regular backpropagation is that BPTT sums the errors at each time step [14]. Figure 1 can help visualize this. The error at time steps h_{t+1}, h_t, and h_{t-1} are summed together and used to adjust weights.

Long Short-Term Memory (LSTM) is an RNN architecture that allows the model to learn long-term relationships in sequential data. The way LSTMs accomplish this is with the use of memory cells in the hidden layers. These memory cells have a forget gate, an input gate, and an output gate. Forget gates determine what information from the previous hidden state is to be remembered or forgotten. The input gate determines what information from the current input is important to remember. The output gate uses the state of the memory cell to determine what information to pass on.

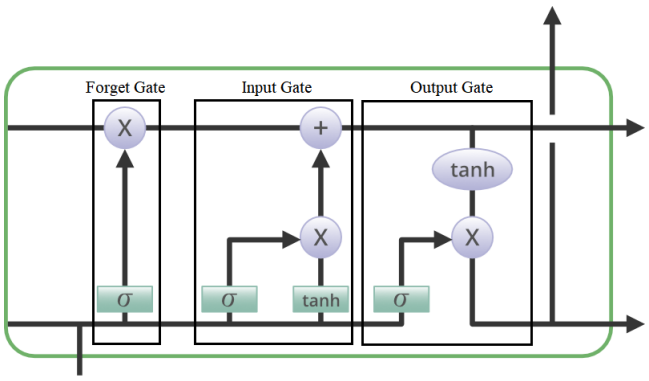


Figure 2. Long Short-Term Memory Cell [7]

[7] This setup allows for finer control over the influence of data items (past state, the input, and the output) on the next memory cell. This allows the network to maintain memory across different inputs on sequential data [2].

As seen in Figure 2, the input and output gates use *tanh* as a function. *tanh* is used to create a normalized vector of the values between -1 and 1. All three gates use *sigma* functions to determine if values are of use. *sigma* functions normalize the input values between 0 and 1 in the case of LSTMs [7]. For the forget gate, information of less importance has values closer to 0, and information of greater importance is closer to 1 [11]. The more important information is remembered, and the less important information is forgotten by the memory cell. In the input gate, the sigma normalized data is multiplied against the tanh vector to obtain useful information about the input. In the output gate, the tanh vector is run through the sigmoid function. This is done to determine what data to output and send as input to the next cell. [7]

Additionally, in Figure 2 the lines and arrows that are outside of the memory cell represent connections to other nodes or hidden states within the RNN.

2.3 Deep Recurrent Neural Networks using LSTM

A DRNN is similar to an RNN, with DRNNs having multiple hidden layers to process each item in a sequence. DRNNs allow for the learning of complex feature relationships within a dataset [2]. In the context of this paper, DRNNs are used to classify sequential data.

2.4 Support Vector Machines and Linear Discriminant Analysis

Support vector machines (SVMs) and linear discriminant analysis (LDA) are methods used to classify data.

SVMs find the optimal hyperplane that maximizes the distance between classes in a binary classifier [12]. Like a straight line separates data in a two-dimensional space, an n-dimensional hyperplane separates data in an $(n+1)$-dimensional space. Figure 3 is an example of how an SVM

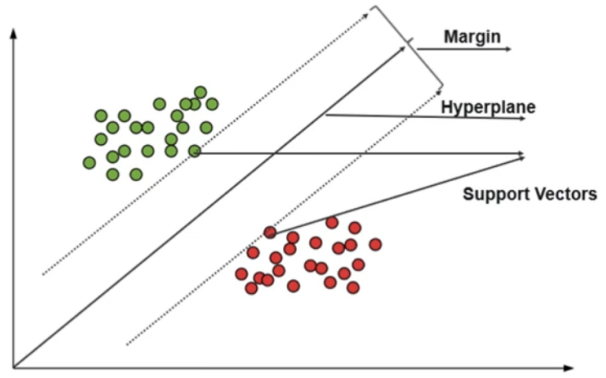


Figure 3. Support Vector Machine demonstrating the hyperplane as a line separating two classes [16]

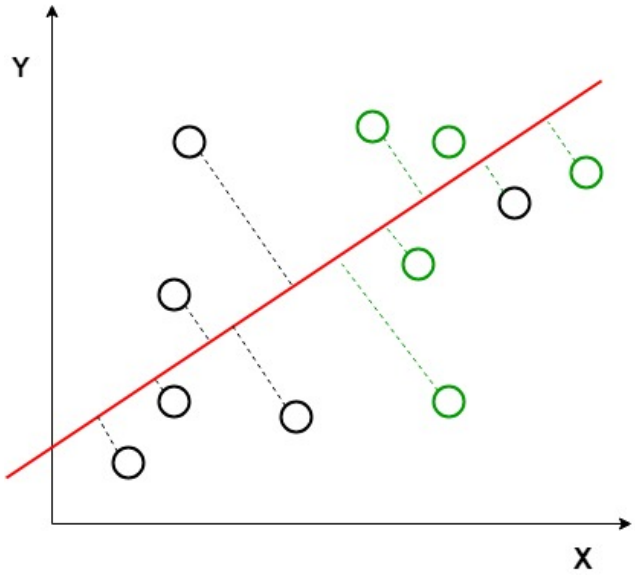


Figure 4. Data Projected onto a 2D space before LDA [8]

works. The hyperplane separates the two classes of data where the margin between the two is the largest possible.

LDA is used when a dataset has multiple classes to classify the data into. LDA accomplishes this by finding combinations of features and the labels of the data to maximize the distance between the mean of the classes and to minimize the variance within a class. The data is then projected onto a one-dimensional space and classified. [8]

In Figure 4, the red line is where the mean between the two classes is maximized, and the variation within the classes is minimized. The data points are then taken and projected onto a one-dimensional space for classification.

Figure 5. Data projected onto a 1D space after LDA (adapted from [8])

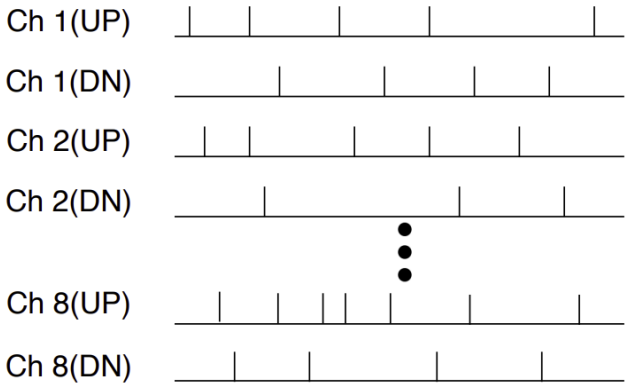


Figure 6. Spike Encoded Data where each spike signifies a point in time where [5]

3 Spiking Neural Networks

Spiking Neural Network (SNN) design can vary greatly. There are many different methods of implementing the different parts of an SNN. In this paper, we will focus on SNNs as applied to prosthetics.

SNNs use sequential data as input. However, unlike the DRNNs described earlier, SNNs use sequences of discrete events (or spikes) as input [15]. This requires data to be encoded to represent where, in time, the value of the data reaches a predefined threshold. These are known as *Spike Trains*. For example, Garg et al. [5] calculate the amplitude of an electromyographic signal across time, and if the difference between two consecutive time samples' amplitudes exceeds a predefined threshold, a spike is labeled at that time. They split the signals into two different inputs: one for upward spikes and one for downward spikes. An abstract example of what this spike-encoded data looks like can be found in Figure 6. Each vertical line is a spike that represents where, in time, the difference in amplitude was significant.

The spike-encoded data is fed into a network of interconnected nodes called a *reservoir*.

Reservoirs behave like RNNs in that nodes can output back to themselves as well as their connected nodes [5]. There are various methods of training the weights on the edges of an SNN reservoir [15]. In Garg et al. [5], the weights are adjusted so that the branching factor of each node reaches a target value; in the case of Garg et al., this value is one. At a branching factor of one, when a node spikes, it should result in the node effectively only outputting to one other node

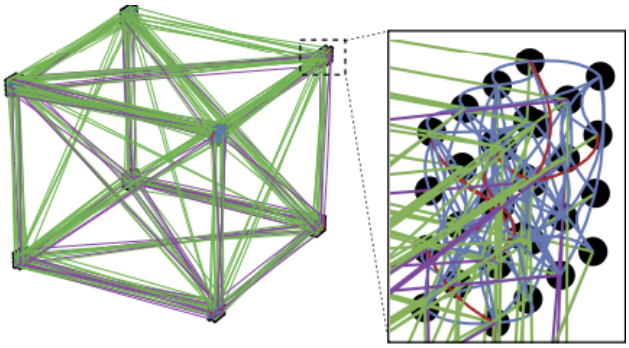


Figure 7. Reservoir Topology used in Garg et al. [5]

even with other connections [5]. This is known as *critical plasticity*.

The reservoir's topology (or shape) is important. The connections between any two given nodes are random, with the probability being based on the distance between the nodes. For example, in Garg et Al. [5], a small-world-like topology is used. The nodes are arranged in the way seen in Figure 7. The different color edges in Figure 7 represent different types of connections. The details of these connections are outside the scope of this paper, for details see Garg et al. [5].

In SNNs, the nodes are inspired by biological neurons. The nodes in SNNs do not typically output immediately. Rather, a node takes in input and updates its internal state until it reaches a threshold. At this point, the node then fires off a signal to the next node, known as *spiking*, and resets its threshold to a predefined base value. These thresholds only adjust as data is fed through the network. As a node receives more input, the threshold needed to activate the node is increased. These are known as Leaky Integrate-and-Fire (LIF) nodes. The opposite is true for LIF nodes as well. The less a node is activated, the lower the threshold to activate the node becomes.

Once the input data has been processed by the reservoir, the state of the nodes is taken as a vector. This vector is then input into a classifier, for example, SVM or LDA [5]. The classifier outputs the class of the gesture the prosthetic user intends.

4 Data and Results

4.1 The Datasets

Surface Electromyography measures the electric activity of a muscle's response after being stimulated by a nerve through the skin. Both machine learning methods discussed in this paper use surface Electromyographic (EMG) signals as input data. Surface EMG is often used to measure a person's muscle movement intentions as it is non-invasive and cost-effective. An option for measuring EMG signals is the Myo armband, a commercially available device. The Myo armband has eight sensors that are capable of measuring EMG signals through the skin. Both methods discussed use datasets collected with the Myo armband. [2, 5].

The datasets used for the two methods are different. The DRNN uses a dataset consisting of four hand gestures (Okay, Rock, Paper, Scissors) and a dataset consisting of seven gestures (Resting hand, Clenched hand, Wrist flexion, Wrist extension, Radial deviation, Lunar deviation, Palm extension) [2]. The SNN uses a Roshambo dataset consisting of three gestures (Rock, Paper, Scissors) [5] and a dataset of five gestures named "Sensor fusion (EMG)" (Pinky, Elle, Yo, Index, Thumb) [4, 5].

4.2 Result Metrics

The metrics used for the DRNN are defined for binary classifiers. When used for multi-class classifiers, the metrics are broken up into individual classes. For example, a data set with three classes, "rock," "paper," and "scissors," is broken into three binary classifier problems; "rock" or "not rock," "paper" or "not paper," "scissors" or "not scissors."

The number of data points labeled as a class and classified correctly as that class is known as the True Positive (TP). For example, when looking at the TP of the class "rock", the data points labeled as "rock", and are correctly classified as "rock" is the TP. The number of data points correctly classified as not belonging to a class is the True Negative (TN). For example, when looking at the TN of the class "rock", the data points correctly labeled as not rock is the TN. The number of data points that were incorrectly classified as a different class is known as the False Negative (FN). For example, when looking at the FN of the class "rock", data points labeled as "rock", but are incorrectly classified as not "rock" is the FN. The number of data points that are not labeled as a class but are incorrectly classified as in that class is known as the False Positive (FP). For example, when looking at the FP of the class "rock", data points labeled as "scissor" but are incorrectly labeled as "rock" is the FP. [9]

The results for DRNN with LSTMs are evaluated using precision, recall, and f1-score as metrics. In multi-class problems, it is important to remember that these metrics are calculated as one class versus all other classes, resulting in a score for each class in a dataset. For example, using the dataset of the DRNN, "okay", "rock", "paper", and "scissor" will have individual precision, recall, and F1-scores.

Recall is the TP divided by the TP plus FN.

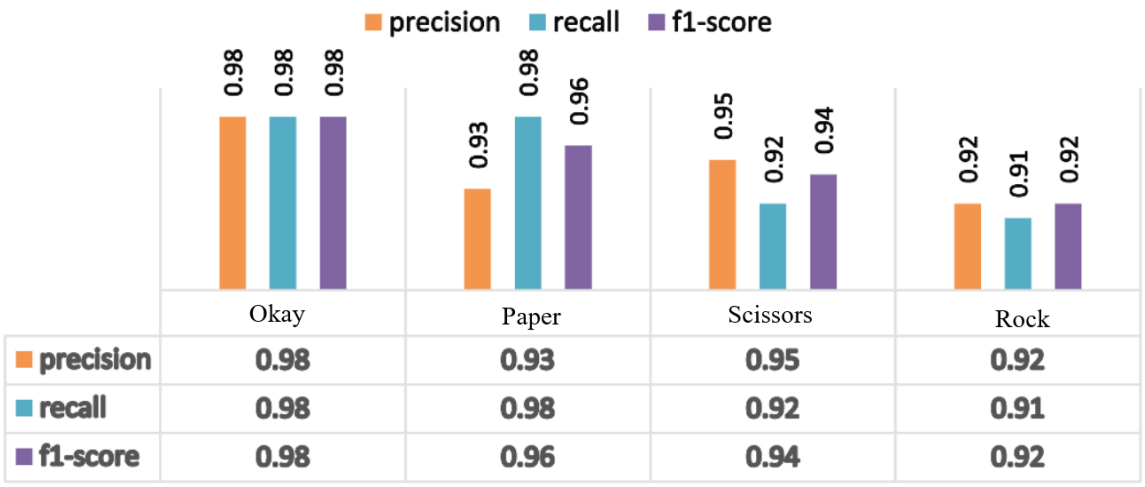
$$Recall = \frac{TP}{TP + FN}$$

Precision is the TP divided by the TP plus FP.

$$Precision = \frac{TP}{TP + FP}$$

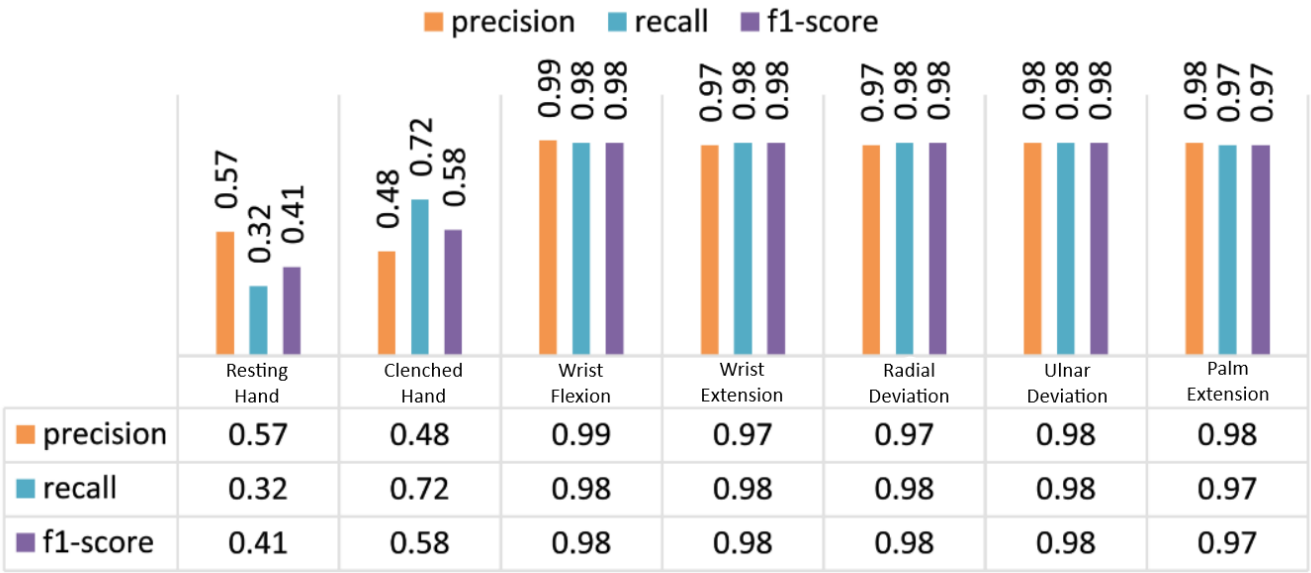
F1 Score uses both recall and precision.

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall}$$



	Okay	Paper	Scissors	Rock
precision	0.98	0.93	0.95	0.92
recall	0.98	0.98	0.92	0.91
f1-score	0.98	0.96	0.94	0.92

Figure 8. Results of the DRNN trained on the four gesture dataset adapted from [2]



	Resting Hand	Clenched Hand	Wrist Flexion	Wrist Extension	Radial Deviation	Ulnar Deviation	Palm Extension
precision	0.57	0.48	0.99	0.97	0.97	0.98	0.98
recall	0.32	0.72	0.98	0.98	0.98	0.98	0.97
f1-score	0.41	0.58	0.98	0.98	0.98	0.98	0.97

Figure 9. Results of the DRNN trained on the seven gesture dataset adapted from [2]

From Alimam et al. [2], the multi-classification model's results for the four-gesture dataset are shown in Figure 8. The DRNN does quite well with this dataset. It's able to classify the "okay" gesture very well, with a score of 0.98 in all three metrics. The results of the seven-gesture dataset are shown in Figure 9. The DRNN doesn't perform quite as well with this dataset. It struggled to classify the resting and clenched hand, with scores of 0.72 or lower. On the rest of the dataset, it scored in the high 0.90s. Unfortunately, Aliman et al. [2] do not explore why certain gestures perform worse than others.

The SNN from Garg et al. [5] uses accuracy as its metric. *Accuracy* is the total number of correctly classified data points divided by the total number of data points [9].

$$Accuracy = \frac{Correct}{Total}$$

The results of the SNN used by Garg et Al. can be found in Table In Figure 10, the results of the Roshambo dataset and the sensor fusion dataset are shown. Additionally, Garg et al. [5] experimented on the datasets using other SNN methods. These other methods are outside the scope of the paper, but it should be mentioned that the SNN looked at earlier in the paper (Reservoir with critical plasticity) performs the best overall. On the Roshambo dataset, the SNN has an accuracy of about 88 percent. The SNN doesn't perform quite as well

Dataset	Method	Readout classifier	Accuracy ($\mu \pm \sigma$)
Roshambo	Spike Enc. & Eval. Baseline	LDA	74.61 ± 2.44%
		SVM	85.44 ± 0.75%
	Reservoir (No plasticity)	LDA	76.44 ± 1.80%
		SVM	81.83 ± 0.93%
	Reservoir (CRITICAL)	LDA	83.06 ± 0.92%
		SVM	**88.00 ± 0.29%**
Sensor fusion (EMG)	Spike Enc. & Eval. Baseline	LDA	53.03 ± 1.38%
		SVM	65.18 ± 3.28%
	Reservoir (No plasticity)	LDA	55.97 ± 2.07%
		SVM	62.33 ± 2.05%
	Reservoir (CRITICAL)	LDA	61.96 ± 2.99%
		SVM	**70.60 ± 3.65%**

Figure 10. Results of the SNN [2]

on the sensor fusion dataset, with only an accuracy of 70.6 percent.

5 Conclusion

The two neural network methods can't be compared because they use different metrics. Looking at the scores of the DRNN, it may be tempting to conclude that the DRNN is better as it has higher scores. However, when calculating Recall and Precision, incorrectly classified data is not taken into account when looking at a specific class. For example, when looking at the recall for "rock", if a "scissor" is incorrectly classified as "paper," this isn't reflected in the score for "rock." This error will eventually be taken into account when calculating scores for "scissor" and "rock." Since accuracy is a measure of correctness on the dataset as a whole these incorrect classifications are always accounted for. Also, the different data sets have different numbers of classes. When a data set has more classes, it adds more complexity to classification, which can result in lower scores [1].

Did one method do better than the other? It's difficult to determine an answer due to differing metrics and data. It may be that both methods are similar in ability. DRNNs expand upon more common NN methods, such as RNNs and LSTM, while SNNs offer a more unique approach. Both methods offer promising ventures into improving the recognition of gestures using EMG readings though. Whether these methods are improved upon or new methods are used, further research into the recognition of gestures using EMG readings can lead to more advanced and lifelike prosthetics.

Acknowledgments

I would like to thank Elena Machkasova, my senior seminar advisor and professor, for all the advice and feedback throughout the writing process of this paper. I would like to thank my external reviewer for taking the time to provide feedback. Lastly, I would like to thank all the people who helped and supported me through the writing process.

References

[1] Evidently AI. n.d. Accuracy, precision, and recall in multi-class classification. https://www.evidentlyai.com/classification-metrics/multi-class-metrics Online; retrieved November 4, 2024.

[2] Hajar Y Alimam, Wael A Mohamed, and Ayman S Selmy. 2024. Deep Recurrent Neural Network Approach with LSTM Structure for Hand Movement Recognition Using EMG Signals. In *Proceedings of the 2023 12th International Conference on Software and Information Engineering* (Sharm El-Sheikh, Egypt) *(ICSIE '23)*. Association for Computing Machinery, New York, NY, USA, 58–65. https://doi.org/10.1145/3634848.3634851

[3] C3.ai. n.d. What is a Classifier? https://c3.ai/glossary/data-science/classifier/ Online; retrieved October 29, 2024.

[4] E. Ceolini, G. Taverni, M. Payvand, and E. Donati. 2020. EMG and Video Dataset for sensor fusion based hand gestures recognition. https://zenodo.org/records/3663616 Online; retrieved November 13, 2024.

[5] Nikhil Garg, Ismael Balafrej, Yann Beilliard, Dominique Drouin, Fabien Alibart, and Jean Rouat. 2021. Signals to Spikes for Neuromorphic Regulated Reservoir Computing and EMG Hand Gesture Recognition. In *International Conference on Neuromorphic Systems 2021* (Knoxville, TN, USA) *(ICONS 2021)*. Association for Computing Machinery, New York, NY, USA, Article 29, 8 pages. https://doi.org/10.1145/3477145.3477267

[6] Geeks For Geeks. 2024. Introduction to Recurrent Neural Network. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ Online; retrieved October 20, 2024.

[7] Geeks For Geeks. 2024. Understanding of LSTM Networks. https://www.geeksforgeeks.org/understanding-of-lstm-networks/ Online; retrieved October 22, 2024.

[8] GFG. 2024. Linear Discriminant Analysis in Machine Learning. https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/ Online; retrieved November 4, 2024.

[9] Google. n.d. Classification: Accuracy, recall, precision, and related metrics. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall Online; retrieved October 24, 2024.

[10] Amber Henson. 2021. Introduction to Myoelectric Prostheses. https://www.armdynamics.com/upper-limb-library/introduction-to-myoelectric-prostheses Online; retrieved November 3, 2024.

[11] Gavin Hull. 2022. Building a Neural Network Zoo From Scratch: The Long Short-Term Memory Network. https://medium.com/@CallMeTwitch/building-a-neural-network-zoo-from-scratch-the-long-short-term-memory-network-1cec5cf31b7 Online; retrieved November 24, 2024.

[12] IMB. 2023. What are support vector machines (SVMs)? https://www.ibm.com/topics/support-vector-machine Online; retrieved September 18, 2024.

[13] IMB. n.d. What is a Neural Network. https://www.ibm.com/topics/neural-networks Online; retrieved October 3, 2024.

[14] IMB. n.d. What is an RNN? https://www.ibm.com/topics/recurrent-neural-networks Online; retrieved September 19, 2024.

[15] Vladimir Lyashenko. n.d. Basic Guide to Spiking Neural Networks for Deep Learning. https://cnvrg.io/spiking-neural-networks/ Online; retrieved November 11, 2024.

[16] Premanand. 2023. The A-Z guide to Support Vector Machine. https://www.analyticsvidhya.com/blog/2021/06/support-vector-machine-better-understanding/ Online: retrieved Oct 21, 2024.