“Seamless” Control: Employing Machine Learning to Predict Patient Movements

Presented by Jarren Berdal & Ryan Hartnett

Engr 859: On - Device Machine Learning
Spring 2024
Prof. Zhuwei Qin

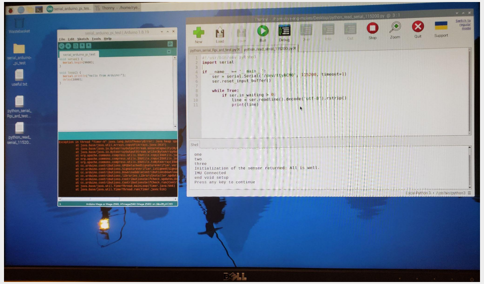
Motivation

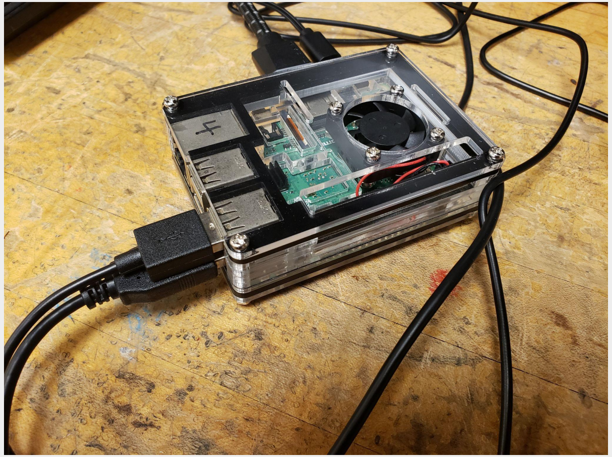
- 15 million people in the world suffer from a stroke year around
  - 80% experience motor dysfunction and need rehabilitation
- Traditional Rehabilitation assisted by physiotherapists
  - Time consuming
  - Limited hospital human resources
  - High cost treatments
- Robotics Devices
  - High engagement and produce + health results
  - High training intensity
  - Repeatability
- Move towards open source
  - Flexibility
  - Adaptability
  - Affordability

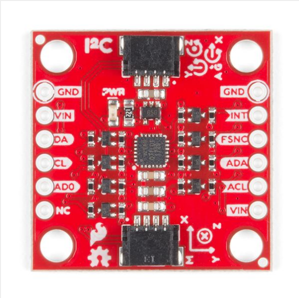
An example of a stationary rehabilitation robotic system (Picture by Yeecon Medical)

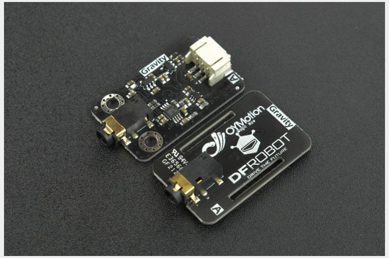
Hardware and Software

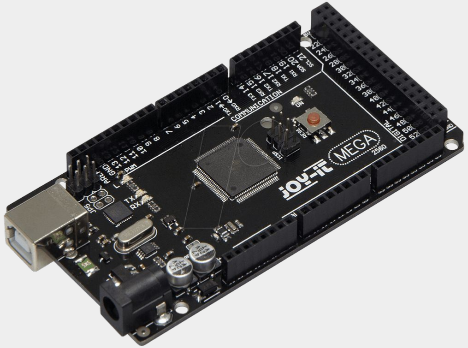
- Raspberry Pi 3B+
  - USB Serial Connection (CoolTerm)
  - Python
  - Pytorch
  - Thonny
- Arduino Mega 2560
  - Sparkfun ICM-20948 Inertial Measurement Units (1x6)
  - DFROBOT Gravity Analog Surface electromyography (sEMG) (3x1)







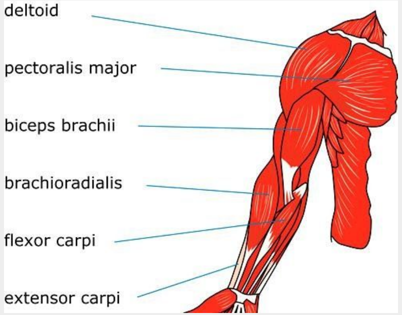




Data Collecting Procedure

- Activity: Drinking a Water Bottle
- Measurement Units (XYZ Ang Pos & Acc)
  - Deltoid
- DFROBOT Gravity Analog Surface electromyography (sEMG) (3x Muscle Action)
  - Flexor Carpi
  - Bicep Brachii
  - Deltoid
- Arduino Mega 2560
- 5 Phases: 1) Rest 2) Reaching 3) Drinking 4) Setting Down 5) Reset
  - Per Phases:
    - 5 seconds each (749 samples)
    - 12 Trials

```
void EMG_ONE() {
for (int j = 0; j <750; j++)  // about 230 samples per second 150 per second  1050 7 seconds
{
  /*
  // For single time stamp at end
  //Serial.print(",");
  if (j == 749)
  { //Serial.print(micros());
  //Serial.print(",");
    timeStamp = micros()- timeinit;
    Serial.print(timeStamp);;
  } else {
  Serial.print(",");
  }
  */

  //Time stamp each line
  //Serial.print("Time Stamp");
  //Serial.print(",");
  //delayMicroseconds(500);

  //int timeStamp = 0;
  timeStamp = micros() - timeinit;
  Serial.print(timeStamp);
  Serial.print(",");
  //*/



  IMU_ONE();
  Serial.flush();
  //IMU_TWO();
  Serial.flush();
  //Serial.print("EMG:Forearm Bicep Shoulder");
  //Serial.print(",");

  for (int analogPin = 0; analogPin < 3; analogPin++)
  {


    // timeStamp = micros() - timeStamp
    int Value = analogRead(analogPin);

    // filter processing
    int DataAfterFilter = myFilter.update(Value);
    //int envlope = sq(DataAfterFilter);

    int envlope = Value;
    int envlope_one;
    int envlope_two;
    int envlope_three;
    // any value under throhold will be set to zero
    /*
```



Methodology

EMFORMER: EFFICIENT MEMORY TRANSFORMER BASED ACOUSTIC MODEL FOR LOW LATENCY STREAMING SPEECH RECOGNITION

Yangyang Shi, Yongqiang Wang, Chunyang Wu, Ching-Feng Yeh, Julian Chan, Frank Zhang, Duc Le, Mike Seltzer

Facebook AI

- Emformer Model
- Audio to Speech Recognition
- Low-Latency
- Developed by Facebook
- Implemented in Genetics Research
  - Sequence to Sequence Recognition
- Model Availability:
  - Pytorch Audio
  - Pipelines

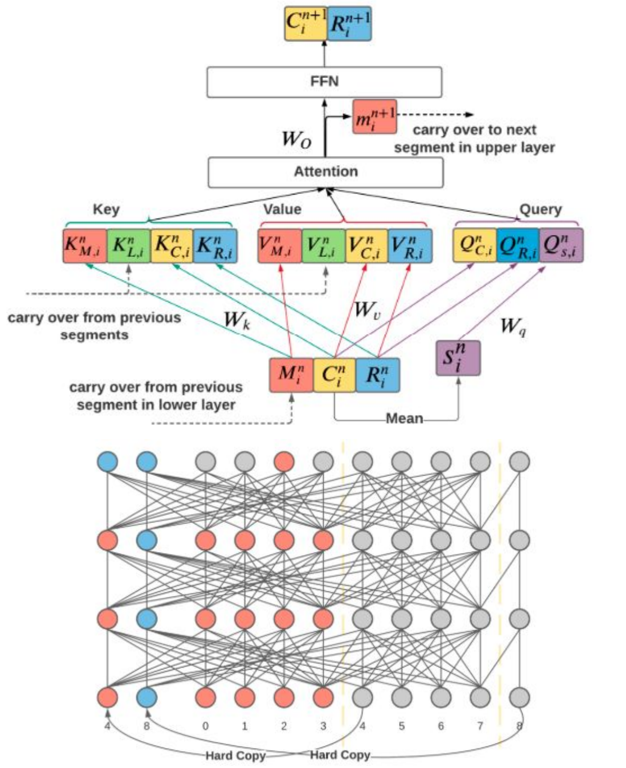
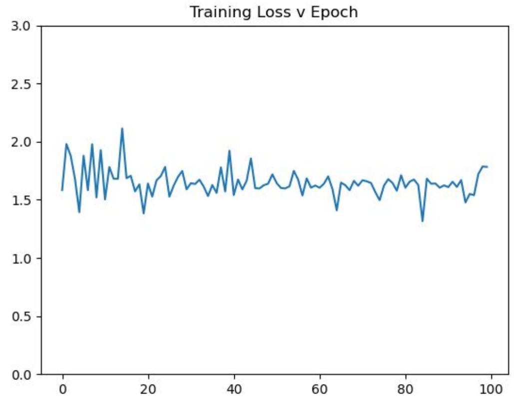
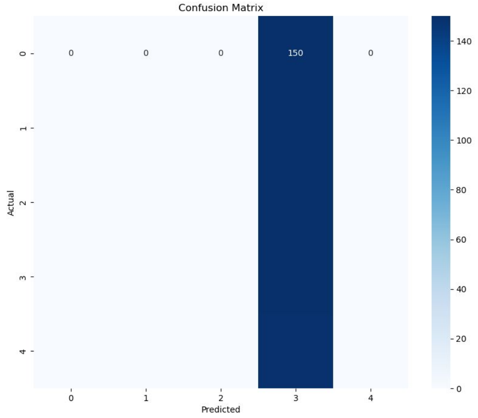


Fig. 2: Illustration of avoiding look-ahead context leaking. The chunk size is 4. The right context size is 1.

Results

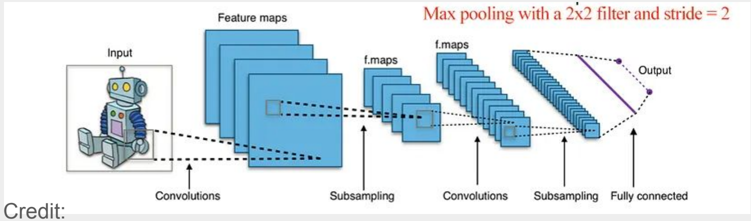
- Emformer Model
  - Complicated implementation
  - Shape mismatches
  - Speech signals tuned to words
  - General waveform computed
  - Long run times
- Learning
  - Multiple tuned attempts
  - Absent
  - Consistent 1 out of 5 classes
  - Roll of dice
- Panic!
  - 2am!





Methodology

- CNN Model
  - Intuitive implementation
  - Shape mismatches
  - Batches based on data
  - General waveform computed
  - Short train times
- Learning
  - Multiple tuned attempts
  - Consistent Improvement

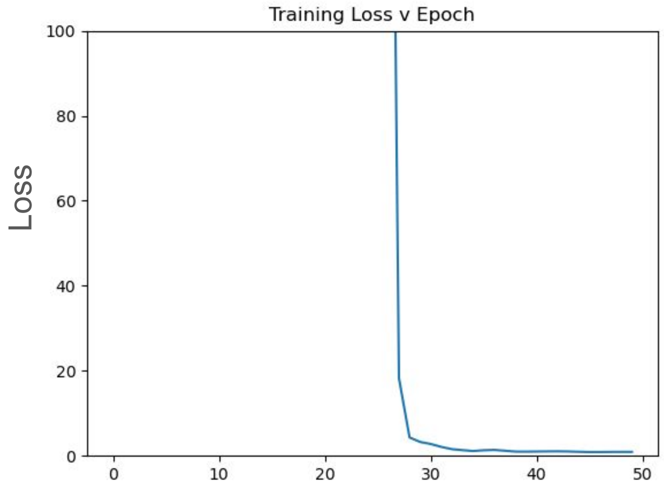


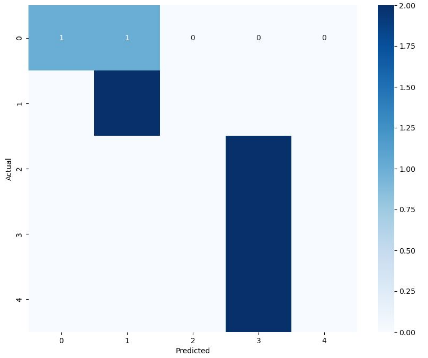
Graphic Credit:
https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118

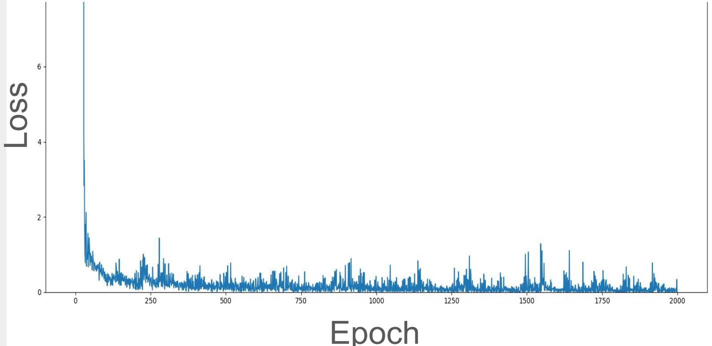
```python
class SimpleCNN(nn.Module):
    def __init__(self, input_channels, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=input_channels, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool1d(kernel_size=2, stride=2)

        # Calculate the size of the flattened tensor
        self.flattened_size = self._calculate_flattened_size(input_channels)
        self.fc1 = nn.Linear(self.flattened_size, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def _calculate_flattened_size(self, input_channels):
        # Create a dummy input tensor with a sequence length of 100 (adjust as needed)
        x = torch.zeros(1, input_channels, 749)
        x = self._forward_features(x)
        flattened_size = x.view(-1).shape[0]
        #print(f'Flattened size: {flattened_size}')
        return flattened_size

    def _forward_features(self, x):
        x = F.relu(self.conv1(x))
        #print(f'After conv1: {x.shape}')
        x = self.pool(x)
        #print(f'After pool1: {x.shape}')
        x = F.relu(self.conv2(x))
        #print(f'After conv2: {x.shape}')
        x = self.pool(x)
        #print(f'After pool2: {x.shape}')
        x = F.relu(self.conv3(x))
        #print(f'After conv3: {x.shape}')
        x = self.pool(x)
        #print(f'After pool3: {x.shape}')
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        #print(f'After flattening: {x.shape}')  # This is mat1
        x = F.relu(self.fc1(x))
        #print(f'After fc1 (mat2): {self.fc1.weight.shape}')  # Print shape of fc1 weights (mat2)
        x = self.fc2(x)
        return x

# Define the input dimensions and number of classes
input_channels = train_data[0][0].shape[0]  # Number of channels in the input
#print(f"Input channels: {input_channels}")

# Create an instance of SimpleCNN
model = SimpleCNN(input_channels, num_classes)
#print(model)
```

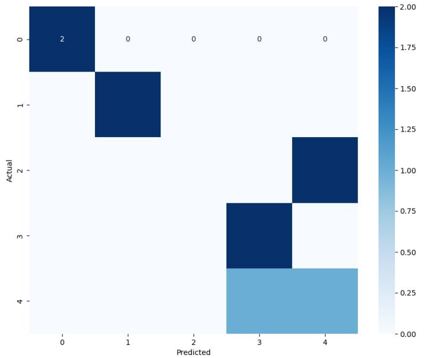
Results

- CNN Model Tuning
  - Test 1:
    - Batch Size: 749
    - Epochs: 50
    - Learn Rate: 0.001
    - Result: 50% accurate
  - Test 2:
    - Batch Size: 10
    - Epochs: 500
    - Learn Rate: 0.002
    - Result: 70% accurate



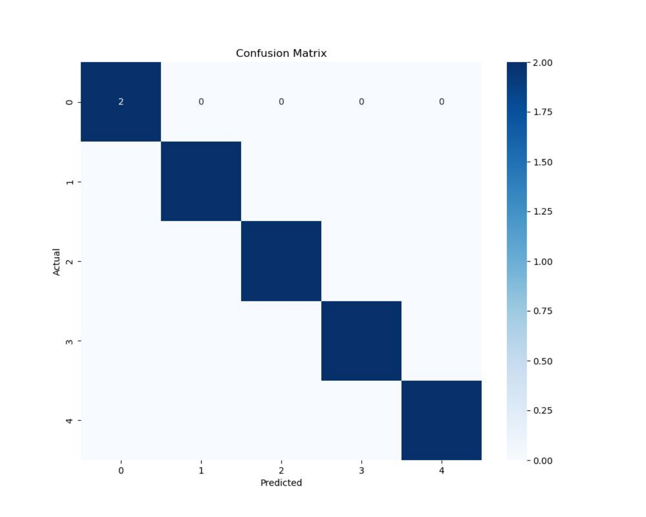


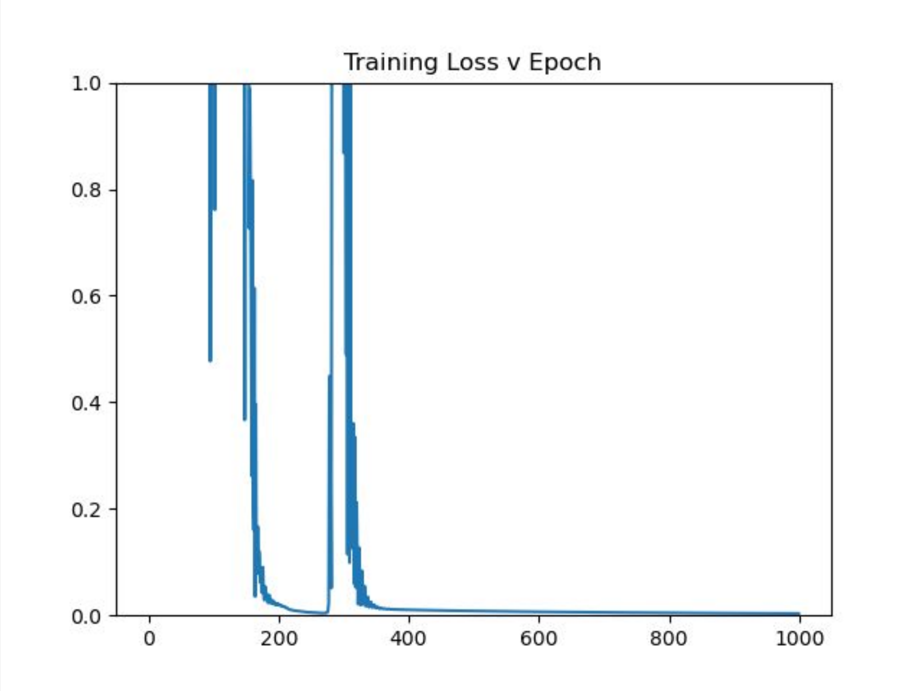




Results

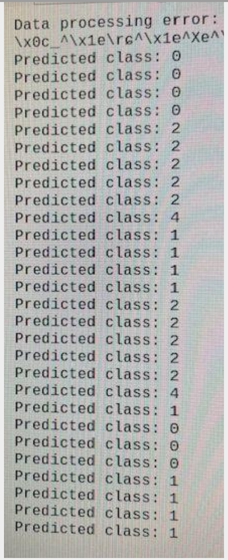
- CNN Model Tuning
  - Test 3:
    - Batch Size: 500
    - Epochs: 1000
    - Learn Rate: 0.0003
    - Result: 100% accurate





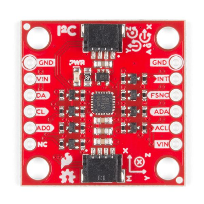
Demo

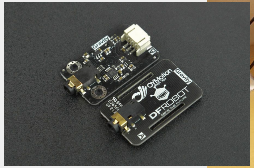
Class #:	Movement:
0	Resting
1	Reaching
2	Drinking
3	Setting Down
4	Resetting

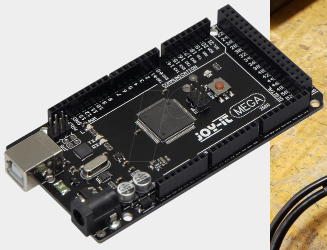


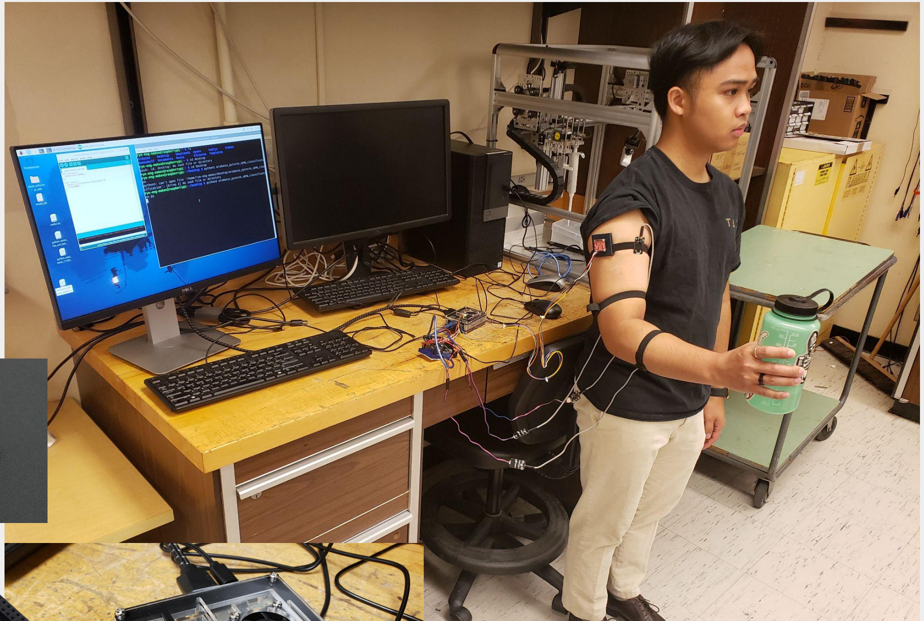
Conclusion

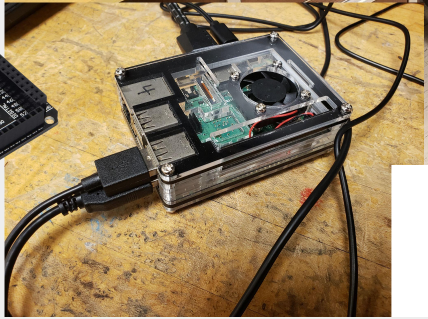
- Model worked! Yay!
- Improvements needed:
  - Increase response time
  - Increase library of motions
  - Fine tune model
- Emformer
  - Interesting
  - May still be viable
- CNN
  - Flexible
  - Intuitive
  - Effective

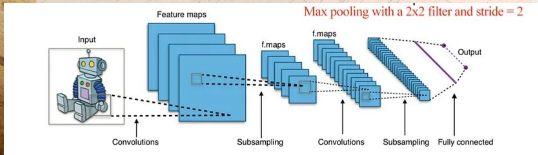












Graphic Credit: https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118