Motion Trace Prediction

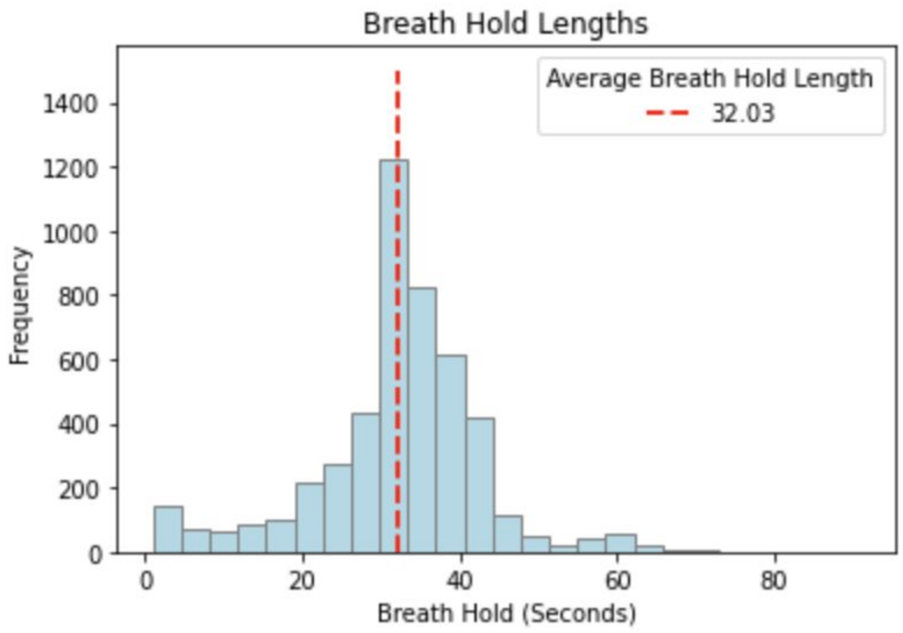
Emma Rainer

Data Preprocessing

- **trace_file_loader.py** has functions to:
  - Return a fully-qualified list of filenames under root directory, sorted alphabetically
  - Equalize trace lengths by adding zeros as padding
  - Determine breath hold start and end indices
  - Create a dataframe given a root directory
- **memory_reduction.py** has a function to:
  - Decrease the memory usage of data frame by changing data types where possible
- **window_data.py** has functions to:
  - Split a list or array into input sequences of len(n_steps) and target sequences of len(n_output)
  - This function is intended to split each trace in a column of respiratory traces and return a dataframe of the windowed data.
  - Reconstruct sliding window traces into full traces.

Information About the Data

- **Average breath hold length:** 32.03s
- **Quantiles:**
  - 25th - 27.94s
  - Median - 32.41s
  - 75th - 37.66s
- **Spread:** 89.94s
  - Max: 90.94s
  - Min: 1.0



Approach 1: LSTM Regression

- **Initial approach:** use an LSTM to predict breathhold length
- **Input:** Trace - Shortened trace (1400 points before start of breathhold, 100 points after start of breathhold)
- **Output:** Data_breath_holds - Breath hold length found using the derivative of the full trace
- **Discrepancies:** Only considered files where the breathhold length listed in the CSV files was within 2 seconds of the breathhold length identified using the derivative of the trace
- **Model:** LSTM Layer + Linear Layer
- **Loss:** MSE_Loss

Approach 1: LSTM Regression Results

- Split data into training (70%), validation (20%), and test (10%)
- Conducted hyperparameter tuning
  - Hidden layers: [10, 15, 50, 100, 150]
  - Num layers{ [1,2, 3]
  - Adjusted learning rate during training
- **Results**: Did not find a configuration of hyperparameters that significantly improved model performance. Model performed worse than predicting the average. Also tried to narrow range of breath holds in the training data to simplify the problem, but did not improve results. Model seems to learn to predict the average only.

Approach 2: LSTM Classification

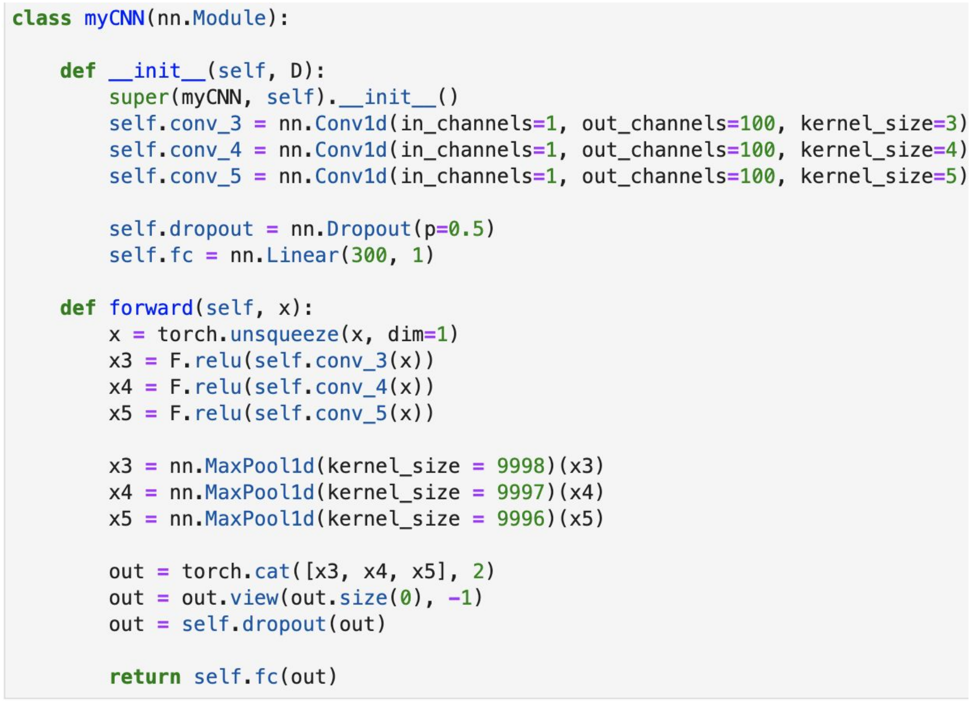
- **Goal:** Binary classification.
  - "Short" breathholds are in the lower quartile of breath hold lengths
  - "Long" breatholds are in the upper quartile of breath hold lengths.
- **Encoded as:** short (0) and long (1)
- **Model:** LSTM Layer + Linear Layer
- **Loss:** binary_cross_entropy_with_logits

Approach 2: LSTM Classification Results

- Loss did not converge
- Model begins to overfit to the training data and validation loss begins to increase
- Attempts to improve results:
  - Used 10th and 90th quantiles to further differentiate between classes
  - Experimented with extremely small training sample (5 short and 5 long traces)

Approach 3: CNN Classification

- CNNs assume a fixed input size so we prepared the data with a fixed size and truncated or padded the sentences as needed.
  - Used length 1500
- **Total Trainable Parameters in CNN:** 1801



```
class myCNN(nn.Module):

    def __init__(self, D):
        super(myCNN, self).__init__()
        self.conv_3 = nn.Conv1d(in_channels=1, out_channels=100, kernel_size=3)
        self.conv_4 = nn.Conv1d(in_channels=1, out_channels=100, kernel_size=4)
        self.conv_5 = nn.Conv1d(in_channels=1, out_channels=100, kernel_size=5)

        self.dropout = nn.Dropout(p=0.5)
        self.fc = nn.Linear(300, 1)

    def forward(self, x):
        x = torch.unsqueeze(x, dim=1)
        x3 = F.relu(self.conv_3(x))
        x4 = F.relu(self.conv_4(x))
        x5 = F.relu(self.conv_5(x))

        x3 = nn.MaxPool1d(kernel_size = 9998)(x3)
        x4 = nn.MaxPool1d(kernel_size = 9997)(x4)
        x5 = nn.MaxPool1d(kernel_size = 9996)(x5)

        out = torch.cat([x3, x4, x5], 2)
        out = out.view(out.size(0), -1)
        out = self.dropout(out)

        return self.fc(out)
```

Approach 3.5: CNN Classification with Fake Data

- **Experiment 1:** Simulated data with breatholds between 500 and 4000 points long
    - Unable to drive testing loss below 0.7
- **Experiment 2**: Simulate data with breatholds of *either* 100 or 9000 points long
    - Training and Validation loss go to 0.
    - This works with the fake data, leading me to believe the model infrastructure is correct for the task, but there is not enough signal to differentiate between similar traces (will not work with the real data)

Approach 3: CNN Classification Results

- learning_rates = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02, 0.05, 0.1]
- Training for 100 epochs at different learning rates did not improve results
- Loss plateaued at 0.68 (essentially the same as flipping a coin)

Approach 4: LSTM Regression and Classification with Equal Trace Lengths

- **Experiment 1:** Equalized traces to 10,000 with random noise at the start of each trace and tried LSTM regression to predict bh length
- **Experiment 2:** Using the equalized traces, tried LSTM classification

Approach 4: Equal Trace Lengths Results

Results: Results were not improved using equal trace lengths

Various Approaches to Seq2Seq Prediction

- Switched task to predicting actually respiratory motion instead of breathhold length
- **Tried the following:**
  - 1, 10,50, 100 step predictions using the full trace
  - Using sliding window approach to create training data
  - Created simulated data
- Mixed results
  - Most successful was predicting regular breathing using a sliding window approach