Non-Invasive Detection of Sudden Infant Death Syndrome (SIDS) through Recurrent Neural Networks

Maximilian Du

Motivation and Problem Formation

I got inspired to do this project when I read about the huge number of infants that fall victim to SIDS each year, and I wanted to create a product that can give parents peace of mind knowing that their infant is being monitored. More importantly, I needed a way to monitor infants in the most minimally invasive method possible. However, I also needed to create something that is easily accessible, deployable, and economic to allow for more usage among all families. With all of these in mind, I created my engineering and coding challenge:

How do I make a non-invasive and widely deployable method to detect and raise alerts when irregular breathing or respiratory arrest happens in sleeping infants?

Existing Ideas

Clip-on Devices

Image source: www.rhmagazine.com

Disadvantages
- Not intrusive, but not practical
- Painful through prolonged attachment
- Might fall off during sleep

Mask-Style Devices

Image source: www.team-consulting.com

Disadvantages
- Hard for infants to wear for prolonged period
- Requires additional equipment
- Optimized for hospitals

Heart Rate Based Inference Devices

Image source: https://vandrico.com

Disadvantages
- Accuracy varies between different infants
- Has to touch skin

EMF Devices

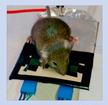
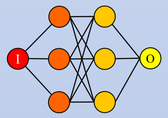


Image source: www.ncbi.nlm.nih.gov

Disadvantages
- Expensive
- Hard to scale up
- Susceptible to interference

What is a Neural Network?

- Inspired by the human brain's neuron structure
- Like estimators for a universal function
- Made of interconnected "neurons" with three parts
  - Input weights and biases
  - Activation function
  - Outputs
- Modeled through linear algebra in the form of matrices



This is a simple neural network called a perceptron

Forward Propagation

Order of operation
1. Multiply input matrix by weight matrix
2. Add result to bias matrix
3. Apply activation function
4. Feed information onto next layer

Activation Function
- Prevents runaway gradients (exploding)
- Makes the network non-linear
- Different types for different purposes
- Recurrents typically use ReLU to prevent vanishing gradient

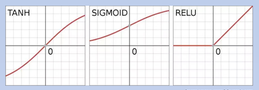
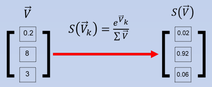


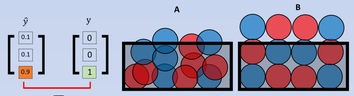
Image source: www.kdnuggets.com

Last node operations
- In my case of classification, the network will return a vector with each value being the certainty of each category
- However, these values are only called **logits**
- This is because their values don't add up to 1, so it isn't a probability distribution
- To turn them into one, a **Softmax function** is used



Loss and Backpropagation (Training)

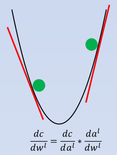
Loss function
- Sees how "bad" the neural network is.
- **Cross-entropy** used for categorical data through one-hot vector comparisons



$D(\hat{y}, y) = -\sum y * \ln(\hat{y})$

$S_a > S_b$ but $S_b \neq 0$. The entropy is relative, thus the name.

Backpropagation (Gradient Descent)
- Traces the error back to each weight and biases
- Once it knows how much each value is contributing to the loss, it tweaks them to minimize the loss
- Takes the partial derivative of the loss with respect to each weight and bias
- The weights and biases are shifted in the direction opposite to the derivative
- Stochastic gradient descent is a more computationally feasible equivalence to direct gradient descent
- **Adagrad** (the optimizer I used) is a slightly optimized version of gradient descent



$$\frac{dc}{dw^l} = \frac{dc}{da^l} * \frac{da^l}{dw^l}$$

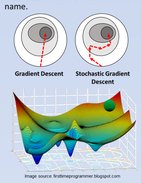


Image source: firstneprogrammer.blogspot.com

Problems of Feed-Forward Perceptrons

- The perceptron is effectively a very powerful high-dimensional function regressor, meaning that **unclean data** is easily **misclassified**
- There is also no **memory**, making sequences like sentences and audio data impossible to decode.

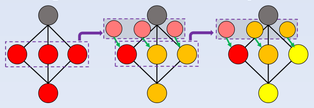
The man walked the dog

The dog walked the man

Recurrent Neural Networks

Design Challenge: How do we give the network memory?

Hidden Recurrence
- The hidden layer is recursively propagated
- Thus, it retains information from all past points
- More recent inputs have greater impact

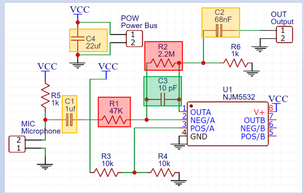


Project Methodology

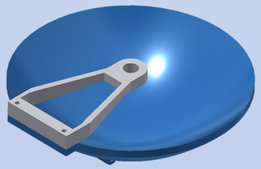
1. Make hardware that can sample faint audio while still being low-noise
2. Record many samples of different inhale and exhales, as well as unknown sounds like background noise and music
3. Create a data feeder algorithm that pre-processes the data for the network
4. Construct recurrent neural network
5. Train on audio samples
6. Apply model for live-streaming operations
7. Develop algorithm to detect abnormalities

Hardware Development

Amplifier circuit development and fabrication
- Used **NJM5532 Low-Noise Operational Amplifier** in an inverting configuration
- 46.8X amplification factor as determined by R1 and R2 (red boxes)
- High pass filters (orange boxes) allowed filtering of nontrivial electrical (C4) and ambient noise (C1, C2)
- Low pass filter (green box) allowed filtering of op-amp-induced noise
- Used surface-mounted components to reduce footprint



Parabolic amplifier design and print
- Design inspired from satellite dish
- Microphone sits at the focal point, collecting all sound that hits the dish
- This allows for the use of general-purpose electret microphones instead of more expensive high-sensitivity ones
- Also allows for the device to be placed further from the subject, making it less invasive



Dataset Making and Splitting

Dataset Making
- Three categories
  - Inhale
  - Exhale
  - Unknown
- 2 seconds of audio each at 8192 samples/second
- I made a program that could record in quick succession

Dataset Splitting and Augmentation
- 2000 samples total (unaugmented), but only 1600 were used
- 30-point isolated test set
- 30 point validation set for progress charting
- 300 training samples per epoch, randomly assigned
- Changes of pitch and added noise were augmentation factors

Data Feeder and Feature Extraction

Preprocessing the data
- Discrete Fourier Transform (DFT) is taken on $\frac{1}{8}$ second long chunks of audio
- This gives 1024 frequencies, which will result in overfitting
- To allow for better generalizing, these are binned into a summary vector with 50 elements
- This is done through using half of an autoencoder which was trained on the same audio samples
- The autoencoder learns how to optimally represent these frequencies in a constricted data space, akin to a principle component analysis

$$F(f) = \sum x(t) * e^{-i2\pi f t}$$

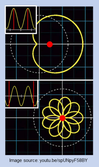
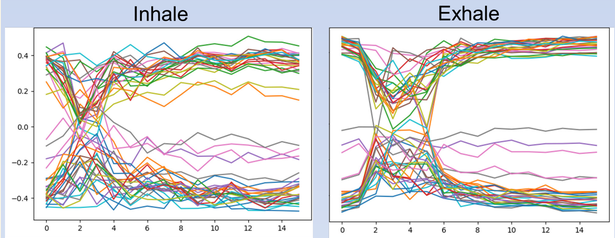


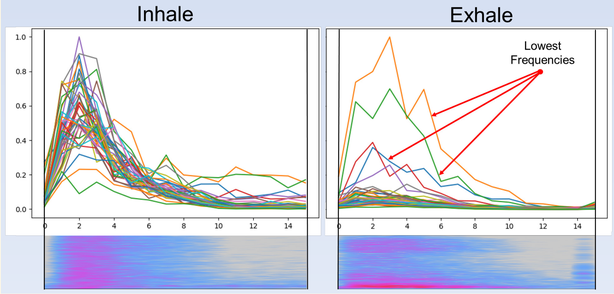
Image source: youtu.be/spt.P8pvF58BY

Network input
- A 50X16 matrix (50 "principle components", 16 timesteps)

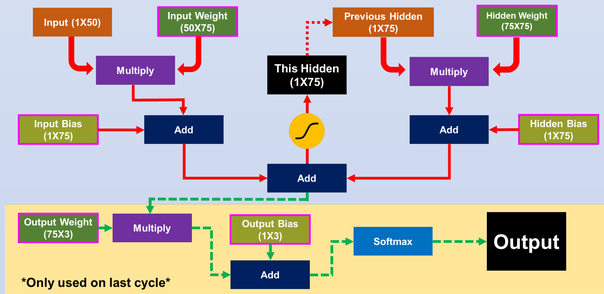
Autoencoder Feature Selection Method



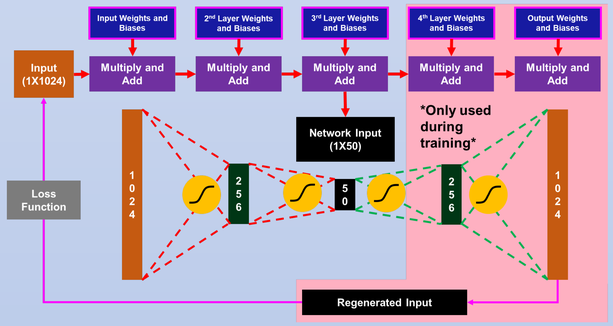
Frequency Binning Feature Selection Method



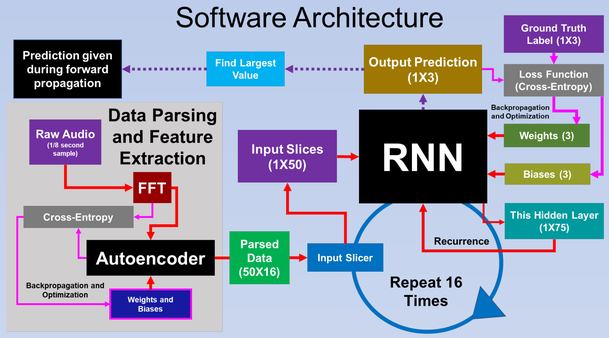
Graph Structure of My Model



Feature Extraction Autoencoder Model

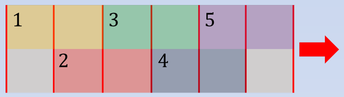


Software Architecture



Running My Model on Continuous Data

Design challenge: How do I use the model on continuous data?
- Use a two-second window that shifts by 1/8 second increments
- This overlap allows at least one total capture of audio
- Predictions are fed into another algorithm that recognizes each distinct breath



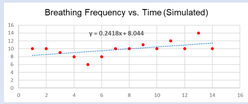
The colored squares represent each time interval that the model uses to make a prediction. Note the overlap.

Design challenge: how do I find each inhale and exhale among the mess of data?
- Many options, but after experimentation, counting occurrences proved most robust
- Detection must last at least **four cycles** at **confidence level $\alpha = 0.90$,** and it must occur **at least 1.5 seconds apart from the last hit**
- This gave maximum accuracy: no false predictions and 6 missed predictions

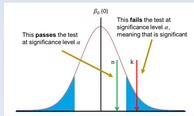
	3.49E-05	1.59E-09	1	
	0.018916	0.000144	0.9809	
	0.203211	0.014045	0.7837	Inflection point
	0.108867	0.861927	0.0292	
1	0.000876	0.997081	0.002	
2	0.00017	0.997127	0.0027	Exhale detected
3	3.74E-06	0.999498	0.0005	
4	8.00E-06	0.943545	0.0564	
5	8.93E-06	0.939034	0.061	
	2.79E-05	0.769225	0.2307	Inflection point
	9.26E-10	1.14E-12	1	
	1.16E-08	1.53E-12	1	

Detection of Abnormalities

Mathematics Challenge: Given a set of numbers, how do I decide if there's a dangerous trend?
- A better way to approach this problem is through a **linear regression t-test for slope**
- Robust and adaptive, as compared to a simple regression or other threshold methods

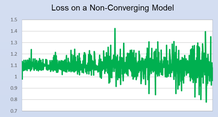


Linear Regression T-Test for Slope
- The slope β follows a t-distribution since we only have a sample of a theoretically infinite population
- Thus, we perform a significance test making $H_0 \rightarrow \beta = 0$
- **If the significance test fails, it means that the data is correlated at confidence level α**

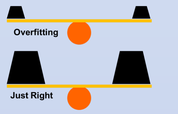


Problems/Challenges Encountered and How I Solved Them

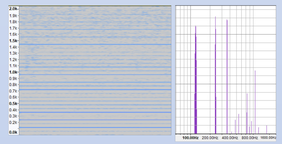
Problem: Network was not converging
- No matter how many epochs were spent, the first network never converged from its starting entropic state
- To debug it, I developed a method to systematically find the culprit
- I intentionally overfit the network with less than 10 data samples
- If the data was the issue, then the network would converge, if not, then the model is flawed



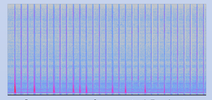
Problem: Network 100% accurate on training, but less than 50% accurate in real life
- Classical example of overfitting
- Confusion matrix showed 100% accuracy
- However, the overall loss for each training point was very low
- This means that the slightest difference could result in a high false positive



Design Issue: Amount of noise introduced by amplifier circuit was non-trivial
- A blank recording from the concentrator showed distinct and sharp spikes at harmonics of 120
- It turns out that the spikes were caused by AC noise, which is 60 Hz, which explains the 120 Hz spikes (zero points) and the harmonics
- To fix this problem, a filtering capacitor with low impedance for lower frequencies was placed across the power bus

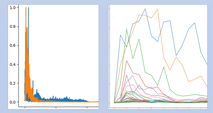


Challenge: Testing and retesting deployment of model in streaming
- It's unreliable and hard to use live samples every time a model configuration needed to be tested
- Thus, I wrote a code that would generate a 5 minute sample of audio with inhales and exhales inserted into ambience noise
- With this program, I could write the exact time of each breath into a ground truth file, making assessment of accuracy much easier



Spectrogram of generated 5-minute audio sample for testing

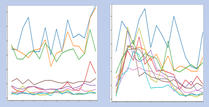
Problem: Inhale and exhale at different amplitudes
- Makes sense, since exhaling pushes air towards microphone
- However, this allowed noise to become as pronounced as the actual inhales and exhales (rightmost graph)
- To solve this, I normalized the fourier transform data before feeding it to the autoencoder
- **Minmax** algorithm was applied to the entire **1024X16** fourier output (makes largest 1, smallest 0)



Normalized FFT of Inhale (orange) and Exhale (blue)

Poorly-Normalized Exhale Spectrum Plot

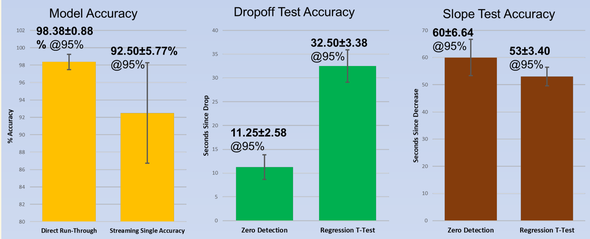
Problem: Data was very noisy
- First microphone I used introduced large amounts of noise
- The spectrum plot for the noise and inhale were consistently very similar, causing false positives
- To fix this, I used a parabolic concentrator paired with a denoising circuit to minimize interference
- **"Garbage in, garbage out"**



Noise

Inhale

Accuracy Assessment



Conclusion

- The sixth version of my model has a very high accuracy: ~92% during streaming, ~98% during pure-sample-runs
- The detection algorithm can detect a simulated stop in breathing within 10 seconds, and simulated dangerous trends within 55 seconds
- Although all of the training data was from me, the existing network could definitely be trained on new data within hours
- Wrote about 3000 lines of code over 6 months

Future Developments

- **Full stand-alone deployment**
- Building of an IOT-style connectivity between Raspberry Pi and smartphones
- **Training on actual infant breathing data**
- Moving away from the statistical test for significance in favor of another neural network
- Using an LSTM for more accurate predictions

ALL pictures and diagrams UNLESS OTHERWISE noted are made or taken by Maximilian Du