University of London
BSc Computer Science
Final Project

Audio Classification with Spiking Neural Networks

Keef Aragon

Introduction

Several years ago, a friend of mine reached out with a good idea for a new mobile app. The app would transcribe vocal music, sung by the user, into instruments and would include a full mobile-optimised mixing studio. After a lot of learning about DSP in general, we had a product that was primarily midi powered. It became clear that even with decent creative-commons sound fonts, the quality of the production from the app was not up to the professional standards we had envisioned. Work that existed was woefully lacking in terms of the quality that we would have needed to achieve. Generated samples were chirpy and had a highly processed synthesised undertone. We realised we needed another approach.Since that time, I've been haunted by what this other approach would be. I wondered if it might be possible to train neural networks to take a transcript of the notes, and perhaps other features from the vocal track recorded by a user, and generate a unique, "noisy," realistic track of the instrument that they were trained to produce. I am proposing a simpler project that introduces some investigation toward this curiosity.

Project Definition

Building an entire model to generate a realistic track of an instrument is something that would be too broadly scoped for me as this programme is my introduction into first hand work with Neural Networks. Instead, I am proposing a project to train a network capable of classifying a stream of audio with a binary classification: "Does the audio currently being analysed contain a particular instrument in the sample."

Very recently, new states of the art have been published for neural networks that do what I pondered was possible. New models have been published that achieve the kinds of results I was originally hoping to achieve, which will be discussed in more detail below.

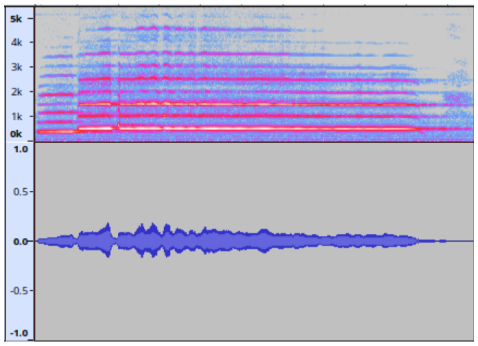
The issue is that some of these models are quite large and would be taxing on an older or lower powered mobile device. Spiking neural networks model the brain by adding statefulness to neurons. They model biological neurons which respond to repeated inputs building axon potential, firing when a threshold is met. As neural networks become more ubiquitous, specialised hardware is being developed to lower the power requirements and cost of running these. One class of such hardware is neuromorphic hardware, which is among the most power efficient and is well poised to model SNNs (Spiking Neural Networks) for extremely low power operation (Neftci, Mostafa and Zenke, 2019).

As I approach this challenge, I will study the applicability and feasibility of SNNs for Audio Signal Processing in this binary classification problem.

Challenges

The first half of the material for “Machine Learning and Neural Networks” (Chollet, 2018) addresses classification of a set of fixed features. However, audio is distinctly time series data. Audio data consists of individual samples that could be likened to the position of a vibrating medium at fixed intervals over time. The richness of the audio comes from the perturbations of the combined signal from a variety of signals all at the same time. And the human auditory system is able to interpret these symbols as they come through (Collins, 2010).

Variable Length Time Series Data



We have well understood algorithms like the DFT and the more frequently used derivative, the STFT, to convert a sequence of time-based audio sample data into a snapshot of an approximation of the data in the frequency domain. While this can go a long way in identifying an actor responsible for a sound, it is still not sufficient. For example, the character of a woodwind instrument like a flute is also a product of the player’s continuous changes of breath over time (Collins, 2010).

So it becomes necessary to model the audio data as a continuous sequence over time, whether in the time domain or the frequency domain.

Signal Noise and Human Perception

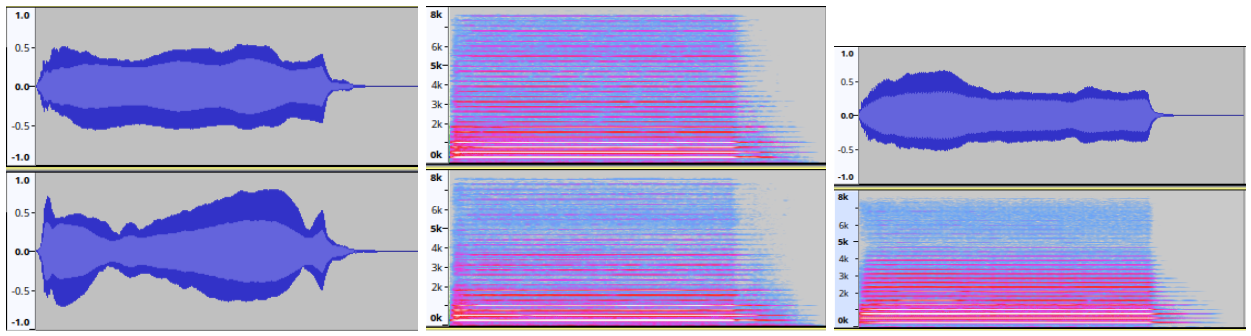
Onset detection is an important task in intelligent audio processing. It may help for prompting an automated response in a sound processing system. For example, a home assistant may wake up and look for a command given a particular prompt: “Alexa!,” “Hey Google!”, “Hey Siri!” And it may be used to classify large collections of audio or video, which becomes important in a world where more video is uploaded to YouTube every minute than a human could watch in a week (YouTube, 2022). It could also be used as an important factor in beat detection for audio applications.

While there are many approaches to onset detection that are used in industry today, in general, these are susceptible to shortcomings. As humans, we have at least some intrinsic ability to pick up beats in patterns of music and identify distinct sounds from samples full of different instruments and events intertwined together. However, this is non-trivial to model mathematically. From the perspective of beat tracking, even the presence of a rich cadre of instruments makes a signal noisy and many modern techniques may still fall short on more complex types of music in spite of working well for simpler genres such as electronic music (Collins, 2010).

Mesaros et al (2010) used hidden Markov Models to detect 61 acoustic events in both clean and noisy samples, achieving an accuracy of 30% and 24% respectively, which among 61 events was substantially better than chance but was still a far cry from that which a human can do trivially.

Intra-class Variability

Another difficult challenge is the level of variability between two samples from the same class. This figure features two samples of the same Acoustic Reed instrument from the NSynth dataset (Engel et al., 2017) playing the exact same note in the Sample and Frequency Domains, along with a sample of a brass instrument playing a similar note:



The two reed instrument samples from the same instrument, on initial inspection, look as different from each other as either does from the brass instrument. Thus we can observe that even within the same class, there is a level of variability for which detection is nuanced and non-trivial.

Background

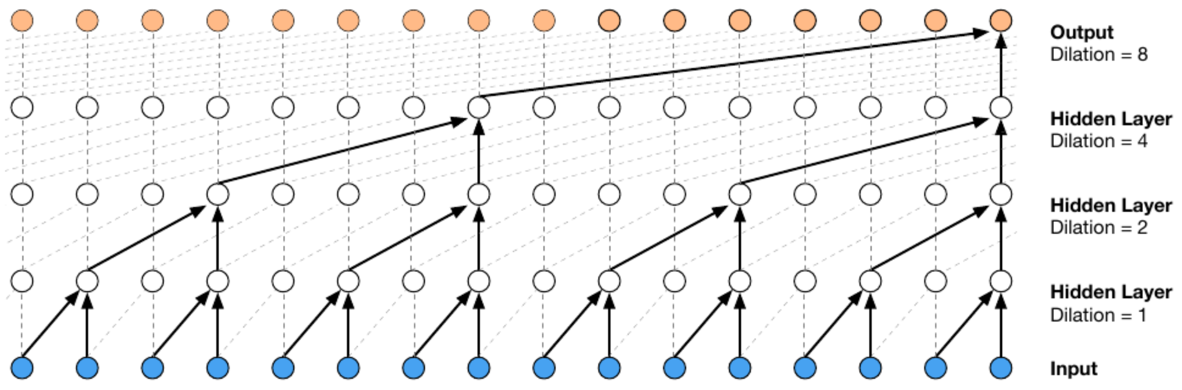
Generative models for instrumental audio

Since my initial motivation for this topic dating back to 2016 comes from a desire to build an application to create instrumental audio with a generative model, I'll first explore a timeline of events from that time until now for this express purpose and then dive into remarkable research orthogonal to that particular problem which still applies to my project.

2016: WaveNet (van den Oord et al., 2016)

While my friend and I were working on our application, the state of the art in generational acoustic models was WaveNet (van den Oord et al., 2016). This was an initiative of DeepMind initiated as a part of the Google Magenta Project. The magenta project wasis an umbrella project which those of us who took Intelligent Signal Processing are familiar with. It specifically sought to address machine learning challenges having to do with audio.

WaveNet, when it was released, was novel because it was an autoregressive model that operates in the time domain, rather than relying on the frequency domain or mel-frequency coefficients. It modelled probabilities of samples through several convolutional layers in a novel architecture deemed dilated causal convolutions. In short, the intuition was that each layer explodes the receptive field of the prior layer while maintaining some statistical relics over a wide time band.



(van den Oord *et al.*, 2016)

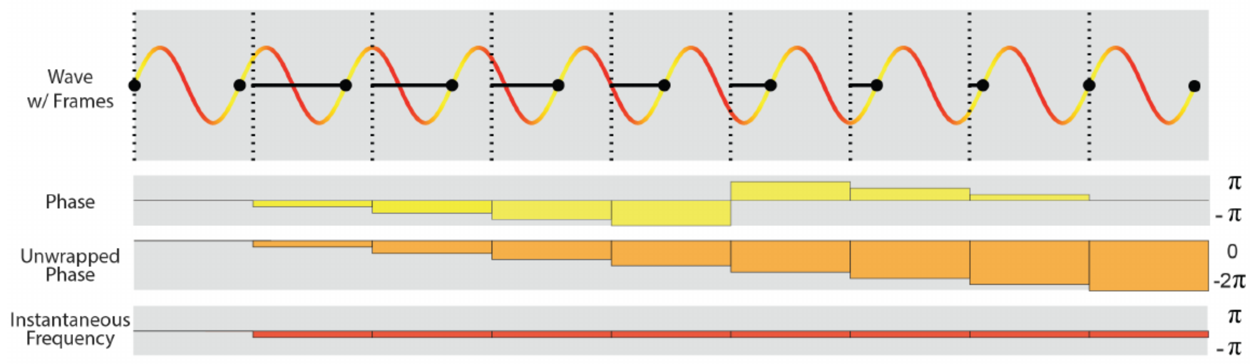
For 16 bit PCM sound, they quantized a softmax layer to 256 outputs, which they found to still be capable of producing convincing audio (sometimes, by my own editorial). Samples of output are available with the proposal video submitted as part of my assignment. Quality of results is mixed with some samples that sound very similar to the original instrument and others that are subjectively hard to relate to their original source.

2019: GANSynth (Engel et al., 2019)

The Google magenta project has continued to redefine the state of the art. In 2019, they published GANSynth (Engel et al., 2019), following the zeitgeist of the time with GANs.

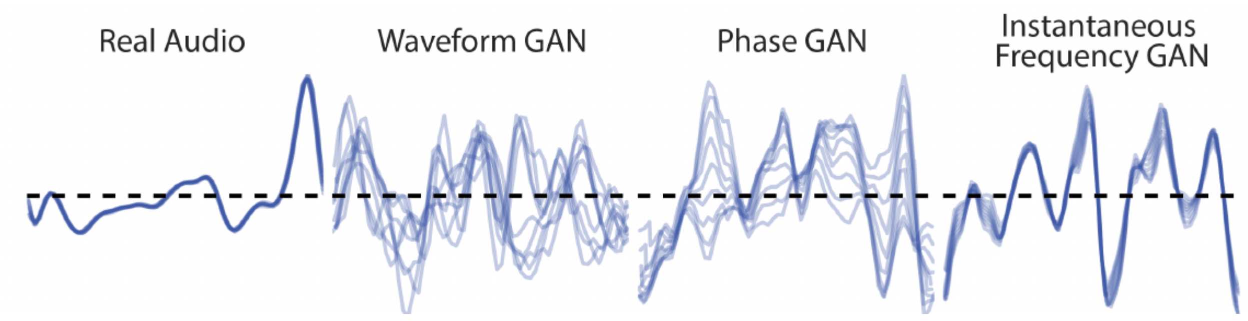
This was another approach to trying to solve the problem that my friend and I were wanting to solve in 2016. That scope of work is beyond what I intend to accomplish in this project, however, the paper is useful for all of the insights it provides for modelling audio mathematically.

In addition to using GANs to address wave generation, Engel et al. (2019) dug into the interplay between the representation of audio and the model architecture. Forgoing a pure convolutional approach for a more advanced representation. This included using mel frequency bands, and frequency domain data after having been normalised to what they call the Instantaneous Frequency. The latter was an interesting approach inspired by the phase vocoder where 2π is added to the signal whenever phase discontinuities are observed and extracting the derivative of the resulting signal.



(Engel *et al.*, 2019)

They used the NSynth data set (Engel et al., 2017), which I also will make use of for this project to train generative GANs using different input representations. They showed that using the IF noted above produces extremely strong phase regularity, which results in far more natural sounding audio, and presumably far less data modelling the same frequency sets for learning. Samples of the output of this model can be found in the proposal video submitted with this project. While GANSynth audio still often diverged fairly significantly from the original signal, it does so in a way that sounds far more natural.



(Engel *et al.*, 2019)

As an aside, the GANSynth paper also directly took on the challenge previously mentioned noting that the model needs to not only model every plausible frequency in the domain but also each plausible phase of each frequency. It was this problem that their IF approach was trying to solve and seemed to do so rather successfully.

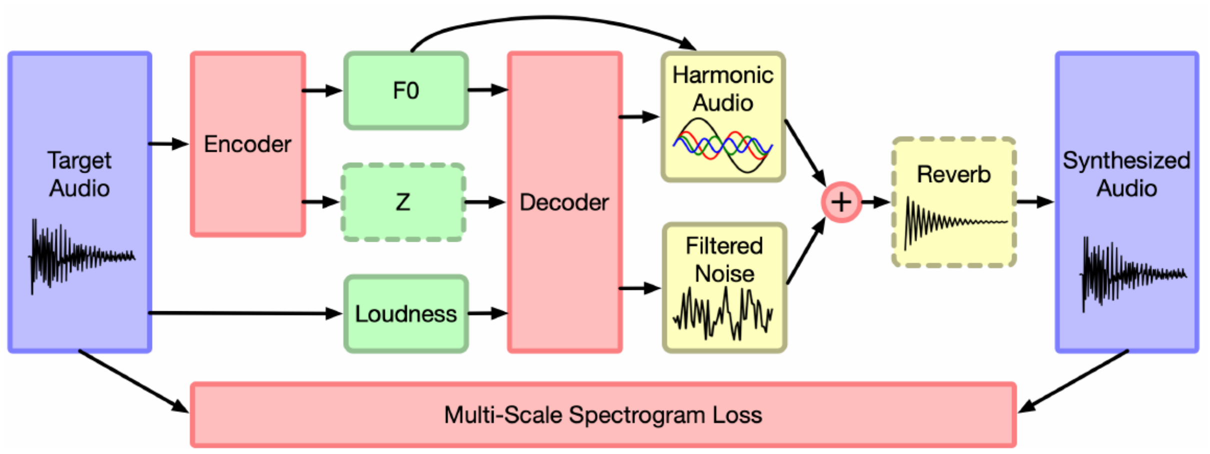
2020: DDSP (Engel et al., 2020)

Again the Google magenta project published a major breakthrough in generative audio networks. In 2020, the DDSP paper was published (Engel et al., 2020) and was the first architecture out of the magenta project to achieve a level of quality comparable to what I was pondering would be possible after abandoning the app project.

Beyond just a paper, DDSP introduced a library of traditional signal processing algorithms that are differentiable and thus usable in modern gradient-based neural networking platforms (Tensorflow). The GANSynth and WAVENET approaches operate in the time and frequency domains respectively. But neither of those approaches exploited the inherent periodicity of audio and the dynamics between the kinetic and potential energy of objects as they dissipate energy during vibration.

Traditional DSP techniques, including the STFT and MFCC used in prior papers, and more importantly oscillators which hadn't been used for similar tasks, exploited this but can otherwise be hard to model as differentiable processes in neural networks. Engel et al. (2020) released a set of tools that solved this shortcoming.

With their library, DSP techniques can be integrated into the neural network as differentiable layers that can be trained with learnable parameters like any other layers.



Using these tools, sounds were produced that were both natural and true to the original samples without large autoregressive models or adversarial loss functions. Samples of the results can be heard on the proposal video submitted with this project.

Spiking Neural Networks

A spiking neural network models biological neurons more closely than traditional ANNs. As we learned in the UofL BSc programme on the AI and ML track, neural networks are really a sequence of differentiable linear equations with a nonlinear activation function. They are inspired by biological neurons but don't seek to emulate them. Hornik, Stinchcombe and White (1989) showed us that artificial feed forward neural networks are linear universal approximators with an asymptotic limit. In practical terms, we can use them to approximate any linear function for which a set of inputs model a set of outputs using mathematical reduction. As approximators with an asymptotic limit toward the "correct" answer, there's an exponential relationship between the desired accuracy and the amount of data required to achieve it. The same relationship exists in terms of the complexity of the model. In practical terms, that means that training an

ANN to a degree of acceptable performance for many real world tasks is extremely hungry for both data and energy. This is an important factor moving forward in the field. The ecological impact of large neural networks has already become a mainstream issue (Toews, 2020).

In terms of application, in ANNs, a “neuron” models a value in a layer, which is generally a term used to describe an n-dimensional vector. Inputs from the prior layer are mapped to some set of neurons in the subsequent layer via a weight matrix. In a feed forward network, there are one or more such vectors with weight matrices along with a corresponding bias vector. Usually when the values for a layer of the vector are calculated, they are also passed through a non-linearity function before being forwarded through the subsequent weight matrix for the following layer. To summarise, while very much inspired by a biological neuron, ANNs are really just sequences of linear equations with differentiable non-linear “hooks” along the way.

SNNs and Biological Neurons

Signalling neural networks (SNNs) are modelled to be much more like biological neurons. In biological neurons, floating point values aren’t passed with every input to every output weighted by some value. Instead, each neuron acts as an independent agent, firing when some preconditions are met. In some ways, ANNs are a higher order abstraction of this process by modelling the concept of a spike frequency train as a scalar value. However, embracing this abstraction reduces the firing rate to an aggregate average. Recent research has shown that biological neurons encode information in the timing of individual spikes rather than simply in an average firing frequency (Vreeken, 2003).

In a biological neuron, signals received via synaptic connections to dendrites raise the action potential of the neuron. When the action potential reaches a threshold, the membrane potential at the axons spikes and a voltage is released, which in turn raises the action potential of neighbouring neurons with synaptic connections to the axons of the firing neuron.

Unlike an ANN, any input to a biological neuron does not cause a regressive output to all non-zero weighted axons. Instead, the action potential of the neuron is increased until a threshold is met, at which point a discrete voltage is released. When this happens, the voltage released is roughly consistent, irrespective of the nature of the incoming signal. That is, a collection of weaker inputs to the dendrites will not lead to a weaker discharge on axons. The spike voltage is generally consistent irrespective of the stimuli that triggered it (*Cosyne 2022 Tutorial on Spiking Neural Networks - Part 1/2*, 2022).

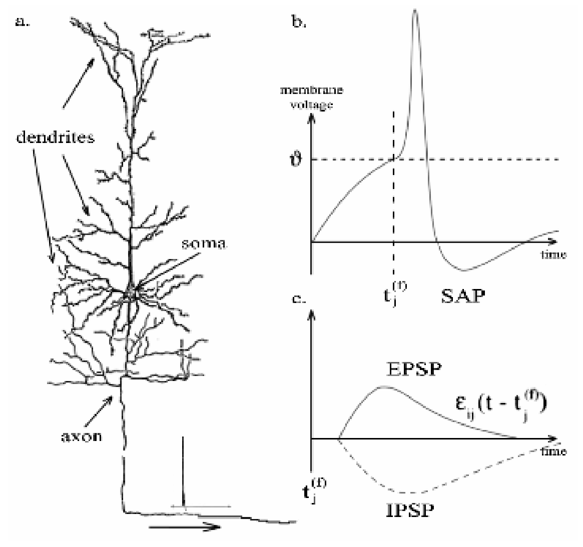


Figure 1. (a) Schematic drawing of a neuron. (b) Incoming postsynaptic potentials alter the membrane voltage so it crosses threshold value ϑ; the neuron spikes and goes into a refractory state. (c) Typical forms of excitatory and inhibitory postsynaptic potentials over time. [8]

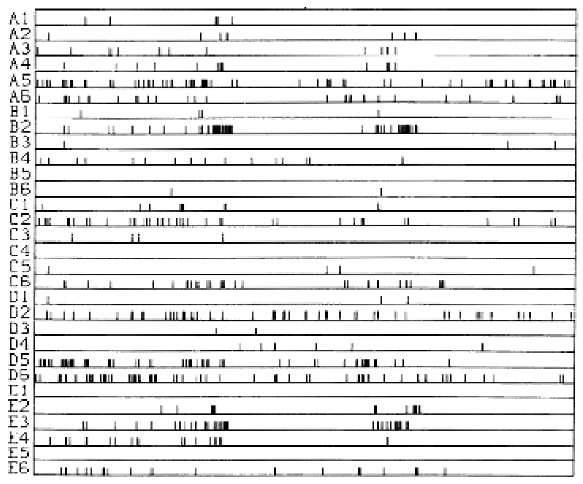
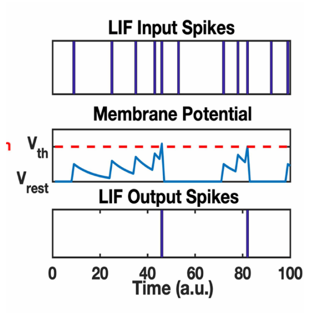


Figure 2. A 4 second recording of the neural activity recording from 30 neurons of the visual cortex of a monkey. Each vertical bar indicates a spike. The human brain can recognize a face within 150ms [24], which correlates to less than 3mm in this diagram; dramatic changes in firing frequency occur in this time span, neurons have to rely on information carried by solitary spikes. [13]

(Vreeken, 2003)

The SNN Model

SNNs model this process by maintaining an internal state that represents the neuron’s action potential. Inputs raise this potential to eventually trigger a binary spike output to the connected neurons. There are different models for the sequence from input to spike but the most common and simplest practical model is LIF, Leaky Integrate and Fire.



An LIF neuron is sensitive to time and exhibits a decay function of the firing potential. Over time, the potential decays but new signals can increase the potential and add to the decayed value. Once the firing potential crosses a threshold, the neuron fires and generates an output signal and resets the internal potential to 0 (Guo et al., 2019).

Efficiency of SNNs

The motivation to approach deep learning processes with signalling neural networks is twofold. On one hand, large scale generative language models have grown to around 1/6th the size of a mouse brain (Zhu et al., 2018; OpenAI GPT-3: Everything You Need To Know, 2021). Very recently, DeepMind claimed that “the game is over” in terms of establishing artificial general intelligence as they unveiled Gato, a DNN capable of performing hundreds of different tasks (Chadwick, 2022). But while this model is much smaller than GPT-3 (but also lacks the NLP capabilities that GPT-3 has), they both share a couple of important characteristics. First, the amount of data that went into training them is astronomical, arguably more than a human being could consume in a lifetime. Second, both training and running the models consumes a tremendous amount of energy.

While not speaking to DeepMind's claims about the game being over, there must be something that we can do in the development of deep neural networks to increase the learning efficiency of models and decrease the power consumption. Early evidence showed that SNNs may require less data for meaningful learning (Payeur et al., 2020; Zheng et al., 2020). And while integrating SNNs into current CPU and GPU hardware typically used for deep learning is actually more expensive, SNNs are well poised to be used on a special type of hardware: neuromorphic hardware.

To discuss the advantage of neuromorphic hardware, we step back and look at the design of memory in modern commodity hardware. We are all aware of the CPU running instructions and RAM that allows storage of temporary values that are calculated by the CPU. This concept serves as the centrepiece to Algorithms and Data Structures I and II, required courses in this programme. To explore neuromorphic hardware, we ignore the design of the bus between the CPU and RAM and focus on the design and challenges of RAM itself.

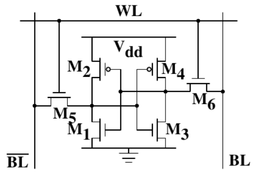


Figure 2.4: 6-T Static RAM

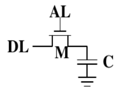


Figure 2.5: 1-T Dynamic RAM

RAM is broadly available in two forms: static RAM and dynamic RAM. Static memory is much faster than dynamic memory but it is also far more complex and more expensive to manufacture as well as more expensive to run. It requires constant power to maintain its state, but for the same reason, reading it is instantaneous and can be done without side effects. Dynamic RAM is simple and relies on a simple transistor and capacitor pair. But it can't just be read continuously. Reading the voltage out of the capacitor dissipates the energy in it and will cause the state to be lost. On top of that, capacitors leak energy at a constant rate. So a capacitor based RAM cell needs to be frequently recharged, often tens or hundreds of times per second. So with DRAM, reading the memory requires waiting for a recharge cycle, discharging the cell and interpreting the charge while recharging the discharged capacitor. This leads to the well known performance cliff that programs face when they have to bypass CPU cache (static memory) and fall back to physical memory (DRAM) (Drepper, 2007).

But when we consider how an LIF SNN cell works, it turns out that much of the trouble caused by the design of DRAM is precisely how SNNs prefer to function. Some input charge is provided in the form of an input spike to an SNN neuron. Over time, that charge decays. Once it reaches a particular threshold, a spike is generated and the action potential is fully discharged. In other words, neuromorphic hardware can exploit the properties of cheap, low-power capacitors to model advanced neural networks. With this type of specialised hardware, SNNs may offer enormous efficiency increases in both data requirements and energy requirements.

History

Spiking neural networks are not a new invention. The earliest models for neural networks were actually much more like SNNs than they were modern ANNs but they weren't practical as general use tools in the way that second generation ANNs would become many years later (Vreeken, 2003). In 2006, (Gütig and Sompolinsky, 2006) proposed a spiking neuron called a tempotron. This incorporated a method to train spiking neural networks based on spike patterns for LIF neurons. These were special-purpose neurons but were cited in the literature for many years and still serve as the basis for SNNs even though more sophisticated learning techniques have been devised.

Tempotrons were a novel way to train spiking neural networks on spatio-temporal spike patterns based on the tempotron learning rule, which was predicated on the formula for an LIF neuron. Having been proposed at a time before differentiable linear equations were the zeitgeist of machine learning, Gütig and Sompolinsky (2006) specialised for the task of binary classification. When a false negative occured, the weights were increased according to a formula:

$$\Delta\omega_i = \lambda \sum_{t_i < t_{max}} K(t_{max} - t_i)$$

which produces gradient descent dynamics with two different loss functions for false positives and false negatives (and no loss for true positives and true negatives). The decay factor is a hyperparameter so that only weights of inputs are learned.

In 2019, Neftci, Mostafa and Zenke proposed a mechanism for using differentiable backpropagation to train Spiking Neural networks. The particular challenge with training Spiking neural networks was that their output is a binary choice. The neuron either doesn't fire, or it fires "at full blast." So when trying to apply gradient learning during backpropagation, the gradient is always either 0 or infinity.

The proposal from Neftci et al was called Surrogate Gradient Learning. Essentially, while performing the forward pass of the neuron, the Spiking mechanisms were used to either produce a 0 or a 1 output. But during backpropagation, a differentiable continuous function is used instead that produces a sharp squeezed sigmoid curve such that a gradient can be calculated. Using this mechanism means that many different SNN architectures are supportable beyond just an LIF neuron. And SNNs can be integrated directly into existing ANN pipelines.

This approach is limited though because it still can fall victim to vanishing or exploding gradients when a large number of SNN layers are used. Other work has since been done to explore the resolution of that problem, some of which will be explored in more detail below.

Related Work

Traditional Neural Networks for Automatic Sound Classification

Mesaros et al (2010) achieved state-of-the-art performance for automatic sound classification as of the time of publication. This work also represented a data set that could be used as a benchmark for ASC tasks. Given the timing, this approach did not involve deep learning. Instead, they extracted MFCC features and used them to produce a Hidden Markov model to label events. The accuracy benchmarks for this approach were far better than chance and represented the state-of-the-art at the time. But accuracy numbers by today's standards were quite low. This work is still worth noting for the historical record and for the use of MFCC features which continues to be prevalent today in the context of deep learning for ASC (Wu et al., 2018).

Not too long after the WaveNET paper discussed above, Hershey et al (2017) published a paper performing the same classification task on 16 classes. They achieved an AUC with their audio data set of 0.930 by modifying ResNet-50 with tweaks to the convolutional layers to accommodate the characteristics of the audio data set. This was a slightly different regression task than the classification task performed by Mesaros. However, using the outputs of the modified ResNet-50 model as input embeddings to a novel densely connected classification model, they were able to achieve an AUC of 0.959.

The process used by Hershey et al (2017) was extremely data hungry using many thousands of hours of audio to train and producing a heavy model for non-continuous inference over a complete block of audio. In that respect, it's hardly applicable for use in this project and represents what might feel like a brute force approach to audio classification with deep neural networks. However, it was valuable overall both because it established a new extremely rich audio event data set from YouTube and because it showed that DNNs could achieve state-of-the-art performance for audio classification tasks. In 2018, Yu et al proposed adding a Multi-level attention model to achieve an impressive AUC of 0.97 over the same dataset used by Hershey et al (2017). This added work doesn't change the core considerations of the work but further shows the viability of deep learning for Audio Classification.

Audio Separation

As a more advanced extension to Audio Classification, some insight might be attained by looking at works of Audio Separation. As noted above, raw audio data is the sum of oscillations in air pressure as they reach the listener. A sound from a single actor, such as a musical instrument, will comprise a myriad of frequencies. The unique sound or tenor of such an actor is the sum of these frequencies, along with how they change over time. Audio Separation seeks to isolate these from one another, in a noisy signal with multiple sound sources. So it's a bit like taking ASC to the next level by determining the delta between the original signal and the signal produced by a single actor. In these schemes, there may be insights into more accurate classification of sounds in a noisy signal.

Hamdy, Vedula and Konduru, (2019) approached this problem with an encoder/decoder model based around CNNs. An auto-encoder was implemented with two convolutional layers followed by a fully connected layer. The decoder was built with several fully connected layers followed by two deconvolution layers per class of audio being separated.They formed a dataset by combining labelled audio data and mixing them into noisy signals and using those as the basis to train. For data representation, they used the frequency domain taking the STFT over 23 millisecond windows. The results were acceptable but not up to par with the state of the art at the time. In spite of the meagre results, this paper is an attractive resource because it represents an architecture that is extremely simple to reproduce with modern deep learning toolkits.

For a more promising set of results in audio separation, we turn to Narayanaswamy et al. (2020). One of the shortcomings they called out of other approaches to audio source separation is that they didn't generalise well due to the limited and inherently biassed data sets that they are fed. Intuitively, if we imagine the task of audio separation of a particular type of audio from a noisy signal, the domain of the audio to extract may be limited. However, the domain of noise from which that signal is to be separated is literally infinite. Narayanaswamy et al. (2020) proposed using an audio-centric pre-trained GAN, WaveGAN in their case, to generate priors from which to train. Interestingly, they used the latent space of the GAN encoder using projected gradient descent to "reverse engineer" the components of the sample by using the latent space of the GAN encoder. The GAN that they used operates in the time domain but the source separation model operates in the frequency domain, incorporating the conversion into the PGD loss function.

Using this approach, they were able to exceed the state of the art in several benchmarks. Unfortunately for this project, the details of the PGD were highly dependent on the structure of WaveGAN and adapting it to something like the models used in the DDSP paper above may be non-trivial and require an expertise far beyond what I could hope to achieve in the context of this project. However, despite the approach being inappropriate to drive this project, there were several key insights to draw from. Particularly, the difficulty of generalisation with straightforward approaches when analysing a noisy signal, a challenge which I will also face.

Using SNNs for Signal Processing

Wu et al (2018) devised a novel framework using SNNs for automatic sound recognition. The pipeline began with a traditional STFT with a hamming window and MFCC based feature extraction phase. This is fed into a "self-organising map," - the novel approach of the paper. The SOM (self-organising-map) had a custom learning scheme where each "neuron" in the SOM was actually a collection of weights that model the input, randomly initialised at first. When a signal passed through them, the neuron with the model that most closely represented the original input signal was updated by having its weights adjusted so as to more closely model the input. The weights from the SOM were densely connected to an SNN layer that learns using the Tempotron method (Gütig and Sompolinsky, 2006). The output spikes from this layer were fed to a binary classification SNN that classified spike patterns as audio events.

This was an interesting approach to a problem I observed in practice while performing some initial experimentation toward this project. It was quite easy to train a traditional densely connected artificial neural-network how to model conversions from the time domain to the frequency domain (endolith@gmail.com, 2018). When performing a DCT or FFT, the intuition is that a set of discrete, evenly spaced frequencies is selected and the signal is modelled against each frequency in such a way that the amplitude of the signal at that frequency is calculated.

This is an approximation though because the actual frequencies present in a signal may not be (almost never will be) perfectly aligned with the intervals selected for the transformation. But another property of these transforms is that the frequencies can be captured near the interval with interference dropping off between intervals at an exponential rate. With a sufficient number of “buckets,” it is possible to reconstruct a raw signal from the frequency domain transformation precisely enough that the difference from the original signal is imperceptible to a human listener.

In my initial modelling, I attempted to first train base layers to perform the FFT on an input sample and then to apply a multilabel regression to get exact frequency amplitudes on a synthesised training set of data composed of frequencies corresponding to model outputs, mixed with random frequencies that were to act as noise. Since the frequencies being labelled were not perfectly aligned to the FFT intervals, the FFT could only approximate the frequencies of the signals that corresponded to output indices. My intuition was that slight adjustments to the weights would allow the neural FFT to adjust the frequency buckets slightly so as to precisely label the amplitudes of the specific frequencies being classified. This intuition was wrong and the model trained as such failed to establish an impressive accuracy.

The SOM model proposed by Wu et al (2018), was an interesting approach for learning these types of classifications. Unfortunately, it was quite novel in the field of neural networks and would be difficult to re-implement in a modern ANN framework. It would therefore be very difficult to integrate with a traditional multi-layer perceptron based model except as a terminal layer or as a separate process called between two distinct models. Though it may be possible to simulate it in a modern deep learning platform using Convolutional layers, doing so is beyond the scope of this project.

The SNN layer was also primarily tempotron-based which exacerbates the challenges noted above. And while it seemed to garner performance comparable to RNNs and LSTM networks, their approach was essentially substituting these two types of layers with an SNN. In fact, in the evaluation of their approach, Wu et al (2018) even swapped out the entire SNN portion of the framework for an RNN and an LSTM. While this makes for a very good evaluation mechanism, it also suggests that they may have failed to exploit the full breadth of the unique qualities of Spiking Neural Networks.

López-Randulfe et al. (2021) had a similar approach applying SNN systems to signal processing tasks. They used a spiking neural network for object detection in radar signal for use in applications like self-driving cars. This is an interesting application because of the implications of running these types of networks on ultra-low-power low-cost neuromorphic chips which would be a requirement for manufacturers in the future world of self-driving cars.

Since they were dealing with radar signals, the process required a two-dimensional DFT since it was a process of correlation where the same one signal could be bounced back at different times with different amplitudes based on the physical presence of objects in front of the radar. However, what is interesting in this paper is that López-Randulfe et al. (2021) encoded the 2D DFT into a traditional linear artificial neural network as I was doing above. After performing the 2D DFT, they did some additional processing to simulate an SNN spike train based on the neural network. The output of this was fed into a SNN-driven CFAR algorithm, which emulates a well known algorithm OS-CFAR, with a spiking neural network. The translation used the simulated spike train to custom train an SNN based on adjusting a threshold for the number of spikes to be received in some delta time given some decay factor.

In reality, none of the work done in this paper was actually done with SNNs. Instead, they used traditional mathematical methods to model a simulated SNN. The findings were very interesting given the real world use case for low-power neuromorphic chips performing important tasks for self driving cars to function. However, there was no indication in either the affirmative or negative that the emulated models they used could be learned in an SNN architecture generically.

Certainly further work should be done to investigate whether an SNN could be trained to do what was described by López-Randulfe et al. (2021). But the current project is more interested in the learning process of Spiking Neural networks for signal processing than it is the execution of them. Ultimately this project is driven by the desire to be able to come up with an architecture that can generically learn to identify an instrument.

As previously mentioned, a still prominent problem with SNNs is that they can very quickly suffer from vanishing or exploding gradients when using surrogate approaches. Zheng et al. (2020) proposed a new mechanism for batch normalisation called tdBN that is specifically made for spiking neural networks. They provided previously established lemmas and mathematical proofs that their normalisation process protected even very deep signalling neural networks from the vanishing gradient and exploding gradient problems.

This enabled them to train very deep SNNs on common benchmark data sets. Using this technique, they were able to outperform SNN-specific state of the art performance on the ImageNet and CIFAR-10 benchmarks. They were also able to establish new state-of-the-art numbers on the DVS-Gesture and DVS-CIFAR benchmarks, which are specifically for neuromorphic approaches due to their temporal nature.

I suspect that this work will continue to play a major role in SNN training with traditional Deep Learning Platforms. Unfortunately, as of the time of writing, it has not become ubiquitous enough so as to find a well-established pre-built package implementing their brand of batch normalisation. As a result, integrating their work into this project will be non-trivial, but may prove to be required.

Looking at image data as an input signal, Kamata, Mukuta and Harada (2021), inspired by the work of Zheng et al. (2020), went beyond just the scope of integrating SNNs into a deep learning pipeline. They created a Variational Autoencoder made entirely of SNN layers.

Variational Autoencoders encoded an input into a latent space. This was trained through a decoder that takes the latent space vector and tries to reproduce the original image from the latent space. This method has been used in other elements of Audio-oriented Deep learning (Roberts et al., 2019).

Using a Spiking Neural network for a Variational Autoencoder presents a specific challenge. The outputs of Spiking Neurons are generally binary. That is, they are either firing or they are not. There's a regressive property in VAE models where the latent space is a continuous area in multi-dimensional space. Modelling strictly binary inputs into the continuous space doesn't work.

More recent advances in VAE architectures use normal distributions in their multi-dimensional latent space. This prevents "holes" and preserves the continuity of the output of the encoder given slight perturbations in the latent space vector. Preserving both, Kamata, Mukuta and Harada (2021) changed the latent space representation as a Bernoulli process modelled over the encoder's output spike train, where such a process modelled the probability bias in a binary sequence. For the encoder, there was a slight cheat where the process of encoding an image from the latent vector used the membrane potential of the last layer of Spiking Neurons, irrespective of the firing state. Doing so may not undermine the benefits of this type of architecture in a neuromorphic device.

To evaluate the SNN process, Kamata, Mukuta and Harada (2021) applied some standard similarity functions of an output image given a latent vector produced by an input image to the original input image. The SNN model broadly outperformed the quality of the ANN by this measure. Qualitative human analysis of sampled images also showed positive results. The major win was that the SNN model was substantially less power intensive, performing 6.8 times fewer floating point additions and 14.8 times fewer floating point multiplications. The advantages in raw electricity costs were likely to be amplified by many orders of magnitude when considering the possibility of running these types of models on a neuromorphic processor, because the traditional ANN would not be possible to run on such hardware.

The results produced by Kamata, Mukuta and Harada (2021) are extremely promising. Though in terms of the quantitative comparison, the baseline was a generic simple ANN rather than the current state-of-the-art VAE ANN architecture. So while the result was promising, it did not achieve any sort of state-of-the-art. However, since it was an SNN network from end to end, it represented an approach that could attain results at a tiny fraction of the cost of traditional ANNs, and thus may present real value, even if only at mediocre quality.

In this project, I am neither aiming for or expecting to achieve state-of-the art performance. So in that sense, it is very encouraging. However, a continuous stream of audio data is a very different challenge than a finite encoded image represented as a single spike train. So the utility may be limited for this use case.

Objectives and Approach

The Google Magenta project (van den Oord et al., 2016; Engel et al., 2019, 2020) stands out as achieving state of the art in many subdomains of audio-oriented modelling, within the realm of DSP and beyond. As a humble undergraduate student with limited expertise, exploring new areas of research, I have no expectation that I will be establishing, or even approaching the state of the art in model performance.

Wu et al (2018) was the only case that I could find in the literature of anyone using SNNs for audio classification. However, their approach relied most heavily on their novel SOM approach which is specialised for the task of audio classification and doesn't generically port to neuromorphic hardware. I intend to demonstrate a novel architecture based on standard SNN neurons for doing a simplified audio classification task. Doing so will enable training a model that can be directly run on general purpose neuromorphic chips "out of the box."

I expect that, while I won't be establishing any new states of the art in terms of performance, I will be able to achieve an indisputably strong advantage versus a generic random choice baseline with a relatively simple architecture that could be theoretically run with substantial power savings when on a neuromorphic chip against the state of the art even several years ago.

Submission

For this project, I will be repeatedly invoking the Deep Learning and Neural Networks project template. The end product will be a neural network but throughout the cycle of my project, I will be examining subproblems in the larger space and digging into the learned behaviour of models. This will inform me in a way that can maximise the quality of the final work.

I will be submitting a detailed written report along with all Jupyter notebooks used for training all of the models.

Model Architecture

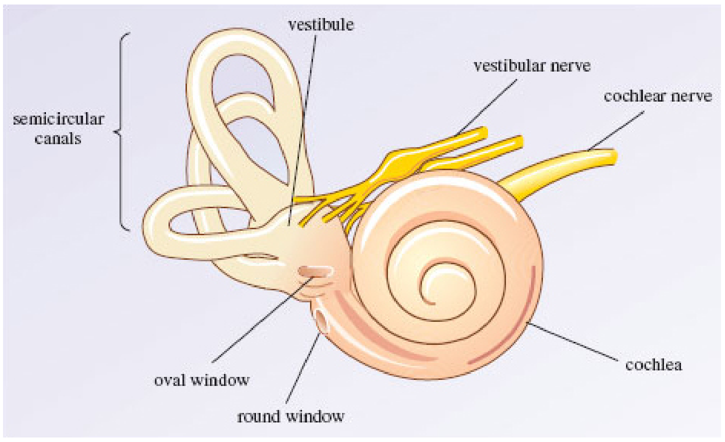
The majority of work on this project will be dedicated to resolving the details of this question. I will be exploring how the model learns from continuous audio signals and will be exploring methods of feature engineering to exploit the learned behaviour to maximise performance. I will revisit this topic later in my final paper with a summative analysis describing the final architecture used. In the meantime, here I explore some of the high level considerations while designing my model.

The Frequency Domain and Biological Plausibility

Some of the literature explored previously was primarily focused on biological plausibility of SNN architectures. That isn't expressly my goal here but I explore the biological foundations of auditory perceptions as a means to inform the ideal architecture for this project. Because SNNs more closely model biological neurons, it is reasonable to assume that additional insights might help from the biological realm. To begin with, it’s valuable to explore the physical systems that enable auditory perceptions in humans, which are derived from similar structures found in other animals.

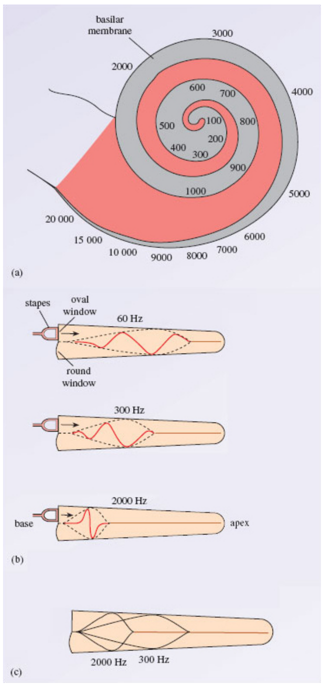
The human hearing system is divided into three distinct components: the outer ear, the middle ear and the inner ear. The outer and middle ear are interesting in the way that they shape the vibrations that make it into the inner ear, including enhancements of key frequencies and modulations that help with spatial resolution of audio (The Open University, 2018; McDermott, 2020). The magnification of signals in the 2-7kHz frequency range is an important consideration from the outer ear. The middle ear is necessary for the conduction of sound from the air filled canal into the fluid filled inner ear. (The Open University, 2018). But it is the inner ear that I lean on the most to inform this project.

The primary structure in the inner ear that functions in sound perception is the cochlea.



(The Open University, 2018)

The cochlea is a spiral structure lined with small sensory hair cells connected to the nervous system that are very sensitive to movements. The area where the cochlea opens to the oval window and the middle ear is the basal end of the cochlea. The end of the spiral is known as the apex. Being a spiral, it’s clear that toward the apex, the width shrinks. The scale of this is such that the basal end is about 5 times larger than the apex end. Also important is that the basilar membrane, which runs along the interior of the cochlea in a V shape, gets progressively less stiff toward the apex. The size difference and stiffness difference means that the frequencies that most affect the basilar membrane vary along the cochlea (The Open University, 2018).



The hair cells that cause spikes in auditory nerves, which ultimately are responsible for audio perception thus are affected differently throughout the cochlea. Responding more to high frequencies at the basal end and to low frequencies at the apex end (The Open University, 2018). Digital audio signals are coded as pressure fluctuations captured in discrete moments in time. But our auditory system actually operates in the frequency domain, with tens of thousands of these small “hairs” that respond to different frequencies.

Discretization in both the time and frequency domain

Because of this, there is no precedent even in nature for transitioning from the time domain to the frequency domain based on spike trains. Biological hearing operates in the frequency domain based on the physical structure of the ear. As a result, I approach this part of audio modelling without the use of SNNs.

However, there still may be shortcomings of traditional approaches for training neural networks to perceive audio in a way consistent with human hearing. The size and softness gradient of the cochlea is essentially continuous. Although there is discreteness at the atomic level, due to the scale, we will consider it conceptually continuous. There are also a finite number of hair cells at various points through the cochlea. These actually serve to discretise the signal in the frequency domain in much the same way that a DCT or FFT does in DSP. However, there is still a problem of scale since there are thousands of these hair cells along the cochlea that are each stimulated by slightly different frequencies.

The corollary in the context of a robust machine learning model is to upsample all input signals to an extremely high quality, but this increases the size of the input signal that is required to capture low frequencies which can lead to reaction delays and performance problems. However, each model in this project is specialising in a particular instrument, or optimistically a few particular instruments. And intuitively, those low frequencies may not need consideration. So one of the concepts I plan to explore if I have capacity, is whether a traditional ANN can be used to learn an optimal SFTFT size.

Technology Used

Throughout this work, I deviate a bit from the Artificial Neural Networks course and the template by using PyTorch as the Neural network toolkit for training my models. Though I found tensorflow to be easier to use and more straightforward for simple tasks, the bulk of SNN libraries and research is based on PyTorch. For my preliminary work dealing with SNNs, I used snntorch (Eshraghian et al., 2022). Another framework that I will look into during the course of my project is spikingjelly (Fang et al., 2020).

Each of the models built will be built in Jupyter notebooks. Supplementary code common to multiple tasks will be included as general Python scripts and imported into the notebooks of interest. Key visual aids and results will be included in the report such that a reviewer need not go through the notebooks in detail except as a validation exercise.

Datasets

In order to train and validate models, I will be relying on publicly available datasets. I will not be recording many of my own positive samples because I am not capable of playing any of the instruments that are being labelled.

NSynth (Engel *et al.*, 2017)

NSynth was mentioned in the background information as a data set used by one of the magenta project papers. This is an extremely large collection of normalised single note audio samples of both real and synthesised instruments. All of the samples are the exact same number of samples and are labelled with both the instrument and the frequency it is playing. In addition to the citation, the actual data set is available for download here.

Other Datasets

Unfortunately the Youtube100M dataset used by Hershey et al (2017) does not appear to be publicly available. Other similar but smaller data sets are available in the research community. One of those is AudioSet (Hu *et al.*, 2020). While I intend to use NSynth (Engel *et al.*, 2017) for a lot of the labelled instrumental data, I will need a wide variety of other data to produce noisy samples to use for training and testing my models.

Noisy Audio

While it may be difficult to find more than a few samples of a particular instrument playing in a noisy environment, this is something that can be synthesised. I will use the various pure audio samples from the datasets I have. I will then programmatically mix these together to produce noisy clips of the instruments I am training toward.

Project Structure

Unlike a user-facing application, I am approaching techniques for modelling audio with a novel approach. The novelty of the approach warrants an iterative approach based on deconstructing and modelling the various subproblems within the domain. Each of these deconstructions will be an exercise in training a neural network of some type. To evaluate the final model, I will use multiple evaluation criteria, primarily based on standard binary classification confusion matrices.

Preliminary work

In my preliminary work before submitting my midterm draft, I trained a black box SNN based model to validate that an SNN driven model could recognise temporal features in audio without any recurrent or convolutional layers. I also trained a traditional artificial neural network to perform a standard FFT. I spent minimal time evaluating this. I only performed a visual inspection of the graphs after performing the fft.

Future work

As I continue to model subproblems and aggregate results, each approach will warrant its own specific evaluation criteria. But throughout the assignment, many of these will be built from a standard confusion matrix with metrics for precision, recall, and accuracy.

Amplitude response

My preliminary work building SNN-based models to distinguish real sounds from their synthesised counterparts were working with extremely sanitised data. Aside from all being the same length, the audio samples were also all roughly at the same volume. This is critical because the inputs that ultimately end up activating SNNs are directly keyed off of the amplitudes of the frequencies being modelled. This will be problematic because the exact same sample at a lower amplitude will cause the neuron to fire less frequently, and conversely, at a higher amplitude, it will fire more frequently.

In future work on the project, this is likely to have a snowball effect as the amplitudes at varying frequencies work in conjunction to trigger the particular sequence of spikes necessary to cause a spike in the next layer. So modulations in the volume will likely have detrimental effects on the model performance. I will need to revisit this problem and model in detail a set of features and a model architecture that is immune to this type of variance.

Recurring patterns in audio

A subproblem to investigate is modelling a chain of Leaky-Integrate-and-Fire neurons to respond to repetitive patterns of frequency changes in the temporal domain. From my preliminary analysis, the modulation of breath from a flute player seems to be an identifying characteristic in instrument identification, in addition to the harmonies that accompany it. For this problem, I will try to model a synthesised repeating pattern in audio with a positive label when the pattern is observed, with varying time shifts in the pattern.

Updating the model to work with noise

Having at least partially resolved the challenges related to amplitude variance, and the ability to pick out and learn distinct repeated patterns in audio, I will need to make sure that the same preliminary model is capable of doing as well with noise. My intuition is that if the other challenges are resolved, adding noise shouldn’t affect model performance too much. However, I need to account for this as a major item of work because that intuition may prove to be entirely wrong.

A learned FFT targeting specific frequencies

The next task will be to revisit the ANN learning to model the FFT. In my preliminary experimentation training an ANN to build the FFT, I tried to get the model to do the next step - to take the result and identify whether a specific set of frequencies were present and at what level. While it was possible to train a model to build an FFT with extremely good accuracy, adding additional layers to perform this next task proved to yield poor results. I think it would be advantageous to revisit this problem to get an ANN to learn to set up the frequency bins to target frequencies of particular interest.

A training pipeline that incorporates noise

As mentioned above, the final model should be capable of receiving a noisy audio sample and identifying whether or not a particular instrument is present in the sample. To train and test this model, I will need to provide noisy samples. I will need to build a pipeline that can generate these noisy samples along with their appropriate labels. To make it realistic, This will probably involve some vocoding of NSynth samples to pack multiple notes in sequence.

Final set of models

Toward the end of the project, I will need to assemble the learnings and code samples from all of the various components to train a series of models that are capable of identifying a set of instruments in audio samples. I will submit a series of models that are each capable of a binary classification for whether the instrument it was trained on is present in an audio sample. I can then package this up into a multilabel classification model that runs each of the instrument models to produce a multilabel classification for each of the instruments given a single audio sample. I don’t plan to use this merged model to train. But it should be viable as a test.

Users

My expectation is that there is minimal utility for the sets of models I produce that can identify instruments. So in that regard, I am not trying to design a user experience for my models in any sense. However, my hope is that this research could be built upon for more use cases. So I am focusing on researchers as my “users.” A device that has a neuromorphic chip running at extremely low power levels could be built to have speech recognition capabilities using the techniques developed here, if carried further. As such, I will put a heavy focus on detailed reproducibility of the subproblems that I model and the final approach. In that sense, my notebooks and my submission will end up being the final product that I hope for my users to get value from.

Project Timeline

	Jun 27	**Jul 11**	**Jul 25**	**Aug 8**	**Aug 22**	**Sep 5**
Midterm Submission	🟦					
Amplitude Response Modelling	🟪	🟪	🟪			
Recurring Patterns			🟦	🟦		
Working with Noise				🟩	🟩	
Training Pipeline					🟨	🟨

Final Models						🟥

The above chart assigns time to the various subproblems that will go into my submission. In starting the assignment, it seemed to me that the scope of work was limited such that failing to complete the project would be low risk. However, after evaluating what work is necessary, it is clear that this isn't the case. As such, it will be necessary to put hard limits according to the Gantt chart to do the required work.

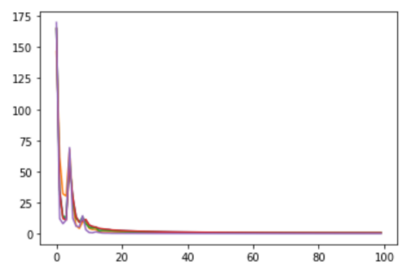
To ensure that I have a submission ready when it's due, I will iteratively work through each of the subproblems, rather than try to accomplish them in full by the end. My preliminary work, both the SNN black box model and the FFT neural network, serve as an evaluation baseline for each of the tasks.

While doing the amplitude response modelling, I will spend the first half exploring the space and building alternate features that can be used as an approach. In the second half, I will be incorporating those features and approaches into a model performing the same task as the first, expecting at least as good results from the baseline dataset, and a maintenance of those with varying volume levels. While working on the recurring patterns, I will take a similar approach, modelling a toy problem first, and then incorporating that approach into the SNN baseline. And the FFT bin-learning can go into the baseline model straight away.

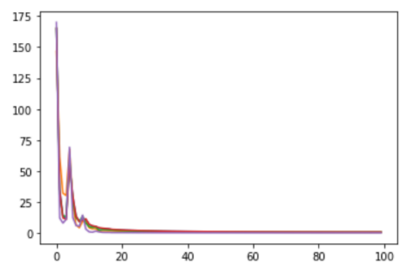
Subproblem Approach and Results

ANN that learns the Fourier Transform (Frequencies.ipynb)

Real Mathematical DFT (166.697395 ms)



Learned equivalent of DFT (175.810393 ms)



Ultimately, a fourier transform can be reduced to a simple linear equation where the frequency buckets are represented as weights in a matrix. My initial intuition was that it should be possible to get an artificial neural network to learn these weights. I first approached this by trying to get a network to learn a single bucket by labelling pure signals of the target and working backwards from there. Unfortunately this didn't work. So I updated that notebook to learn the full FFT by following an example in Tensorflow. In the post where someone had trained a model to perform an FFT, the author warned that it was a toy project and would suffer performance penalties that make it untenable (endolith@gmail.com, 2018).

I actually found this not to be the case. The learned FFT was slower than a hand crafted implementation, but only by a small factor. The process of training was to use a pipeline that simply generated random noise. A standard implementation of the FFT was then applied to the random noise and used as the label for the model.

The model is a single layer with no bias. ANNs don't work well with imaginary numbers so as a preprocessing step in the forward pass, the model splits the input into the sine and cosine of the input multiplied by 2π. The outputs are then trained to produce separately both the real and imaginary parts from the library implementation. This process may be the primary cause of the delta in latency.

For such a simple process, I found the Stochastic Gradient Descent optimiser worked better than more advanced algorithms. The optimal number of epochs to get a loss value within the range of floating point rounding error differed based on the FFT window size. As a default, I used 200 because the process only overfits within that floating point rounding margin for error. Smaller windows would get to a loss value well below 10^{-6} within 20 or so epochs.

This was in part due to a custom loss function. Using a standard Mean Squared Error loss function, the early loss values would be extremely large and would dramatically overshoot the bottom of the gradient. Eventually, this would overflow the precision of a floating point and result in the loss quickly increasing to infinity and never converging. To get around this, I took the mean squared error of the log of the predicted and actual values. The forward pass of the resulting model produced exactly the same output as the hand crafted FFT with a minimal performance delta.

The next phase was to try to use this as a first layer to learn the original task. My thought was that you would have an FFT layer that could continue to learn different frequency bins to accurately label the presence of a particular frequency. Having the ability for a layer that is capable of honing in on the specific frequencies that can be used to identify an instrument was one of the features I had hoped to include in the final submission.

However, this part of my modelling was not as successful. To train and evaluate this model, I picked 9 frequencies, mostly due to them being whole numbers, with most lining up with musical notes. The dataset is then a random set of frequencies added together, usually with one or more of the 9 specific frequencies present. These were added with an amplitude which is also random.

The model takes in raw audio samples which get fed into the aforementioned FFT ANN as an entry layer. There's then a batch normalisation layer followed by a fully-connected hidden layer of 64 neurons which feed into a fully connected output layer. The output layer is tasked with labelling the amplitude of each of the 9 frequencies being tracked.

The loss function is a simple per-element L1Loss over the 9 frequencies of interest. A correct label is one in which the

```
[{'train_loss': 7.71585549004376,
  'valid_loss': 22.093637299537658,
  'accuracy': 0.021180555555555557}
 {'train_loss': 5.428861467353999,
  'valid_loss': 21.945079731941224,
  'accuracy': 0.5609027777777778},
 {'train_loss': 5.409874633140862,
  'valid_loss': 21.911021900177,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.404335951060057,
  'valid_loss': 21.898151755332947,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.401944482699037,
  'valid_loss': 21.891910338401793,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.400779131986201,
  'valid_loss': 21.88889081478119,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.40021761059761,
  'valid_loss': 21.887450957298277,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.3999286102131006,
  'valid_loss': 21.88664174079895,
  'accuracy': 0.6281388888888889},
 {'train_loss': 5.3996448187157515,
  'valid_loss': 21.882446193695067,
  'accuracy': 0.6224027777777777},
 {'train_loss': 5.372188764251769,
  'valid_loss': 21.539423775672912,
  'accuracy': 0.5996666666666667}]
```

labelled amplitude is within 3 decimals of the actual amplitude. Though the loss function takes into account the entire prediction. After trying several optimizers, while none gave results on the scale desired, NAdam seemed to converge the fastest to the best score.

The training process converged on the best accuracy at 3 epochs and minimised validation loss after 8 epochs. The accuracy score as per the above peaked at 62.81%. This was clearly indicative of learning happening. Even a binary identification of the frequencies in the signal would have a chance accuracy of 11% but in this case, the model was also labelling amplitudes. So this actually represented a score that was many thousands of times better than chance.

Preliminary instrument binary classification (MFCC.ipynb)

The end result I am aiming for in this project is a robust model architecture that can be trained to recognize the presence of an instrument in any audio sample using signalling neural networks. To validate this as a valid theory, I wanted to ensure that a simple SNN driven model could be trained to perform a simpler version of this task.

As a preliminary model, I used the NSynth (Engel et al., 2017) dataset without any preprocessing. I specifically selected the samples of real and synthesised flutes. These were pre-normalised and free from noise. The task for the model was to differentiate between the real flute and the synthesised flute. This was a binary classification problem where a synthesised flute garners a false label and the real flute a true label.

This exercise was in large part an attempt to validate my intuitions about the advantages of using an SNN for an audio labelling task. While examples in the literature of audio classification using SNNs is rare, there’s an abundance available describing audio classification with traditional ANNs. Sometimes this is performed with recurrent architectures with or without attention (van den Oord et al., 2016; Huang et al., 2018; Mobin, Cheung and Olshausen, 2018; Yu et al., 2018; Roberts et al., 2019; Engel et al., 2020). Convolutional networks are also common as a mechanism to classify audio (Hershey et al., 2017; Hamdy, Vedula and Konduru, 2019; Koretzky, 2019). Both of these approaches provide a mechanism to deal with the temporal nature of audio data.

However, they tend to come at a computational and electrical cost. A single signalling neuron from an SNN can model simple temporal mechanics and can do so in a way that is compatible with extremely low power neuromorphic chips (Zheng et al., 2020). So part of this preliminary work was testing whether this would hold true for audio data. As such, I implemented a very simple feed-forward network to classify the difference between a synthesised flute, and a real one, at least within the NSynth dataset (Engel et al., 2017).

Model Architecture

I spent many days working on this model: changing architectures, tuning hyperparameters, customising loss functions, etc. I kept a poor record of the various approaches that I tried so here I describe where I finally landed.

The input to the model is in the form of the raw audio stream. It is modelled to accept a single channel sequence of raw audio. However, it accepts batches of audio so multi-channel audio could be passed but each channel will get analysed independently. The first layer to the model is a library STFT of the sample converted to the mel spectrum, along with MFCC features. No research was done into the usefulness of these.

The mel scaled frequency bins and MFCC features are sent to a spiking layer with a learned alpha multiplier on the input signal decayed by a learned beta multiplier. The activation threshold is a constant 1.0 with a reset mechanism back to zero. These are then passed to another fully connected spiking layer of 32 neurons with a learned beta decay multiplier with a subtraction reset mechanism where the membrane potential is subtracted by the activation threshold (1.0). So if a neuron in this layer is highly excited by a pulse, it may repeatedly fire. These are then fully connected to a single LIF Neuron with a constant beta decay rate of 0.99, a subtraction based reset mechanism, and no other learned parameters. The role of this neuron is to ensure that repeated positive labels from previous layers fire and preserve a positive label when one is found.

All SNN layers were built using snntorch (Eshraghian et al., 2022).

Data Set

As previously mentioned, this subproblem was trained and validated entirely on the NSynth dataset (Engel et al., 2017). This data set is several GB in size and will not be included in the submission. It can be acquired here. The examples of real and synthesised flutes were extracted from the archive, with the json descriptors being ignored. Positive labels were derived from the presence of the word `_acoustic_` in the filename. A standard PyTorch dataset based on file globbing for `flute*.wav` was used to model the data set.

The NSynth data is already packaged with partitions between a train, a test, and a validation split. However, this presented some challenges early on. At least for the flute data, the training set is heavily biassed in favour of the real acoustic flute. Whereas the validation and test sets are both heavily biassed in favour of the synthesised flute.

Training procedure

Due to the mismatch in bias in the data, a custom loss function had to be devised. As training progresses, the loss function pushes the model more and more in the direction of biassing toward a negative result. Without this, the model would quickly converge on having a strong bias toward positive labels with train accuracies approaching 100% only to have validation runs match at only 35%. So to unbias the model, it starts paying a miniscule extra loss penalty for guessing a positive label when the input data is a negative example.

As training moved on, that penalty very slowly increased until it reached a factor of 3x. The best performance was found when the factor approached 2x but didn't go much further. Anecdotally, this might have been correlated to the actual bias present in the training data set which has about a 3 to 1 ratio of positive examples vs negative ones.

The loss function hyperparameters are the initial bias, which was set to 0.1, and the inverse decay rate, which is a beta parameter that is applied between each batch. In this case, I landed on 1.001.

The loss function was run on the full sequence of labels applied throughout the span of the audio sample. The frequency domain conversion aligns the FFT window to 100ms of audio. So a 4 second audio clip resulted in an nx40 matrix of feature values and will produce 40 labels. Based on whether the last output neuron was spiking when that 100ms sample was analysed. The model's predicted label was obtained by capturing the mode of the predictions for all of the samples and using that as the full sample label.

Since the analysis being done was temporal in nature, it was expected that often, the label started incorrect but as more samples were ingested, it was corrected. As such, the loss function avoided penalising a portion of the sample. The ratio required to be correct is another hyperparameter for the loss function. In my case, I used 0.55.

In summary, The loss function aggregates the 100ms windows, and applied a mean squared error loss to the full window. It then discounted this loss based on a correctness ratio and applied a relu to this to prevent any negative loss. It then applied a proportional additive penalty to account for the bias difference between the train and test / validation data. And returned the mean of that loss.

The spiking neural network library used supports surrogate gradient learning for the spiking neurons. All of the SNN layers shared the same hyperparameter for the surrogate gradient learning process: a slope of 100. During the forward pass, the spiking neuron fired at 1.0 or 0 creating either a 0 or infinity gradient. During backpropagation, the gradient was modelled as a sigmoid. Other options are available but this worked the best. The library recently introduced ATan (2 days before project work began) but this has yet to be released. Future work will hopefully be able to experiment with this surrogate algorithm. The slope of 100 is the slope of the sigmoid at the exact centre.

In training, I tried optimization with Stochastic Gradient Descent, RMSprop, and several flavours of Adam. For the former two, I may have been impatient but neither seemed to allow the model to converge at all. All of the flavours of Adam converged well but NAdam seemed to converge more slowly but more optimally at least when taken alongside the many other hyper parameters in use. Each training cycle was performed on batches of 25 audio samples. The flute training set contained 8,773 audio samples with 6,362 of those being real flutes, and the remainder synthesised ones. With these hyperparameters, overfitting seemed to happen after the 2nd epoch.

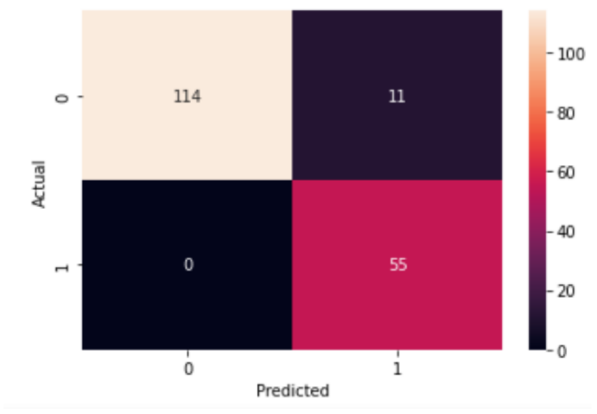
The validation set, which was used for selection of the "best" model contained 470 audio samples. 155 of those were real flutes and the remainder were synthesised. The test set had 180 samples. Only 55 of these were real flutes with the remaining samples all being synthesised.

Results

Between each epoch, the model score was checked against the entire validation set. If the accuracy exceeded the best seen accuracy, that model was saved to disk before the following epoch ran.

The best accuracy score against the validation set was 94.89% accuracy. This was the model that was produced from this process that I evaluated. I went back to try to tune the score even higher but was unable to do so.

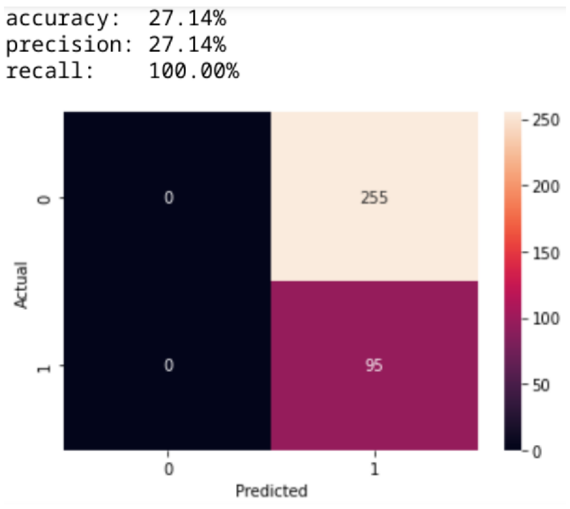
Running that model against the test set rendered an overall accuracy of 93.89% with a precision of 83.33% and a recall of 100%. In spite of a loss function that tried to force some bias into the model to tend toward negative labels, the model still mispredicted 11 of the 125 negative examples as positive ones.



On manual inspection of the 11 mispredicted samples, a majority of them weren't easily explainable by a human listener. That is, my subjective perception was that 8 of those 11 sounded distinctly synthesised.

To validate that the model wasn't just finding key frequency differences between the acoustic and synthesised samples, irrespective of the change over time, I then ran a validation where I picked a random 300ms segment (aligned to a 100ms boundary) from 35 test samples 5 times and ran the same evaluation on the model using only these 300ms samples.

This validated my assessment that the model was picking up temporal features and also elucidated some of the mechanics of how the model learned to identify the instruments.



```
accuracy:  27.14%
precision: 27.14%
recall:    100.00%
```

It appeared that the model's default behaviour is to call any audio a real flute until it learns otherwise over time.

To investigate this, I looked through the learned parameters of the model. Interestingly, it seems that some of the frequencies that are uninteresting for the identification of a flute were given a beta decay factor of 1.0. That is, the activation of some frequencies didn't decay at all. This seemed to have been accounted for by complementary connection weights between layers, with a mix of positive and negative weights and biases, with none of them being 0.

I'm very pleased with the high degree of accuracy that the model achieved. This was a different test than what has been presented previously in the literature. It was without noise and was of standard amplitude, and it was specifically distinguishing between two very similar classes of audio. But I still found it noteworthy that the levels of accuracy achieved were on the same scale as the state of the art when looking at the different problem of sound detection in noisy samples..

The model analysis did reveal some counter-intuitive learnings that the model made, which may be useful in future modelling and training to drive better results. As I progress forward, I want to drill into the learned behaviour of individual neurons and their resulting spike trains to see if I can come up with a model of learning that feels more consistent with human perception.

Dataset Generation

During the process of working on the project, we had a chance to submit a questionnaire that peers could answer as a user test. Incorrectly expecting that this would be a chance to sample a large population, which I wouldn't ordinarily have access to, I wanted to get a sense for thresholds in human hearing. The idea would be that I would generate some noisy audio samples with instrumental music being played at various volume levels, with a single audio source as a control and two additional ones as experiments (we could only ask 3 questions). In order to do this, I had to reprioritise a set of later tasks that would be required to generate audio with which to train and test the final model.

Musical Score Generation

Musical Score Metadata (vae/index.js)

Having learned about MusicVAE (Roberts et al., 2019) in the course CM3020, I decided to use this to generate random musical scores that I could transform into audio samples. As was the case in the course, I found that the python examples no longer work with the latest versions of python and associated packages, so I used magenta's musicVAE.js to generate json files that contain the musical scores. The associated program also could use the computer's MIDI sequencer to play the audio sample before transcribing it to json, using eadymidi (Dinchak, 2022, p.). I used this to generate a small handful of audited samples that felt subjectively like real music and a very large dataset of un-audited samples to use for training and validation of the model.

Conversion to Audio (VAEOutputTransformer.ipynb)

Using these JSON files, I deviated from the process used in CM3020. Using python, I wrote a notebook that reads the json score transcriptions and produces audio by using the huge library of audio samples provided by the NSynth dataset (Engel et al., 2017) along with the DSP tools provided by librosa (McFee et al., 2022) and numerical processing tools from numpy (Harris et al., 2020) to resample and time stretch random selections from the nsynth dataset to produce the rendered audio. The MusicVAE scores were each 8 second samples as per the model. As such, the generated audio was 8 seconds or (occasionally) less, as MusicVAE would occasionally generate pauses at the end of a sample which would be ignored.

Noise Audio (NoisyAudioGen.ipynb)

For noise, I turned to an audio dataset from Google outside of the Magenta Project. AudioSet (Gemmeke et al., 2017) is a dataset of post-processed labels and extracted audio features from a huge collection of YouTube videos. For building a noise dataset, I used the labels to identify a set of YouTube video segments which contain White Noise, Machine Noise, and "Hubbub" - chatter and background noise associated with environments crowded with people. Using a notebook, I then downloaded and converted these samples onto disk.

Merging these together

For building a dataset to use for training and validation, an automated process takes one or more of the noise samples, amplitude-scaled randomly, and overlays them together onto a musical sample, also amplitude-scaled. The whole sample is scaled such that distortion is prevented and clipped.

For building the questionnaire data, this was done by hand with a single generated noise sample at a controlled amplitude being overlaid with 3 different musical samples, each scaled progressively lower. This was done using Audacity (Audacity, 2022).

Frequency Signal Response (VolumeAgnostic.ipynb)

One of the first problems to model is to make sure that we generate a significant enough spike for a frequency at the appropriate volume level at the first signalling layer. To accomplish this, we model an SNN layer that doesn’t learn anything but is instead hand tuned to generate the appropriate signals.

The output of an SNN neuron is a discrete binary signal. It's either “fire”, or not. This is the case with biological neurons as well. There's some minimum threshold to capture a frequency. This may not necessarily be the firing threshold. It is possible that a faint sound may take longer to perceive if the input spike train is infrequent but just frequent enough that the time between the spikes is close enough that it is above the frequency at which the decay function and the increase in membrane potential reach equilibrium.

This might represent the nerves attached to cochlear hairs that carry a signal in the frequency domain. I don't know this to be the case but it is the intuition for the purpose it serves. As such, I define a minimum normalised power threshold to fire and model a firing layer without learned parameters to try to reach this goal.

In Week 14 I did a peer review to try to establish appropriate thresholds, but I was able to review only one other assignment and I had no reviews of my own. So I instead designed this experiment with some members of my family reducing the amplitude of a sine wave signal at 800 Hz by factors of 1/10,000th. And I recorded the minimum amplitude at which the signal could be heard on headphones on my laptop with the master volume at 40%:

Person 1: 0.0002 Person 2: 0.0002 Person 3: 0.0005

What makes this a bit difficult is that I've specifically designed the model to have an Mel amplitude minimum with all of the amplitudes normalised relative to each other after a cut off. I arbitrarily chose a range of 0.05 for this cut off. From there, I examined the Mel Spectrum of the 800Hz signal at amplitude 0.0004 to get a sense of the normalised mel power at this raw signal amplitude.

I found that it was approximately 0.0075. When doing the proportional scaling, this would consider the minimum range at 0 - 0.05. So that 0.0075 would be scaled up to 0.15.

So with the LIF equation:
$t_n = \beta t_{n-q} + I_t$

If we work backwards:

$$t_0 = 0$$
$$t_1 = 0\beta + 0.15 = 0.15$$
$$t_2 = t_1\beta + 0.15 = 0.15\beta + 0.15 = 0.15(\beta + 1)$$
$$t_3 = t_2\beta + 0.15 = 0.15\beta(\beta + 1) + 0.15 = 0.15(\beta^2 + \beta + 1)$$
$$t_4 = t_3\beta + 0.15 = 0.15\beta(\beta^2 + \beta + 1) + 0.15 = 0.15(\beta^3 + \beta^2 + \beta + 1)$$

So we can simplify this by saying

$$t_n = 0.15 \sum_{j=0}^{n-1} \beta^j$$

If we integrate this, we get:

$$\frac{0.15(\beta^n - 1)}{\beta - 1}$$

More generally, for any **constant** input I, we can say the membrane potential at t_n will be:

$$I_n = \frac{I(\beta^n - 1)}{\beta - 1}$$

Up to the spike. Where 0 < β < 1

My 800 Hz sample uses 8 seconds of audio and reduces this to approximately 400 time steps, each of which representing approximately 2 milliseconds. Shelton and Kumar (2010) show that mean auditory reaction time is 284 milliseconds in humans. Kemp (1976) finds that neurological activity in response to audio takes 8-10 milliseconds. So I will attempt to model the spike train such that it fires at least every 64 milliseconds for the faintest of signals and fires much more quickly for strong signals. 64 milliseconds is represented in 32 time steps of our signal so we want to model our threshold and beta such that:

$$1 = \frac{0.15(\beta^{32} - 1)}{\beta - 1}$$

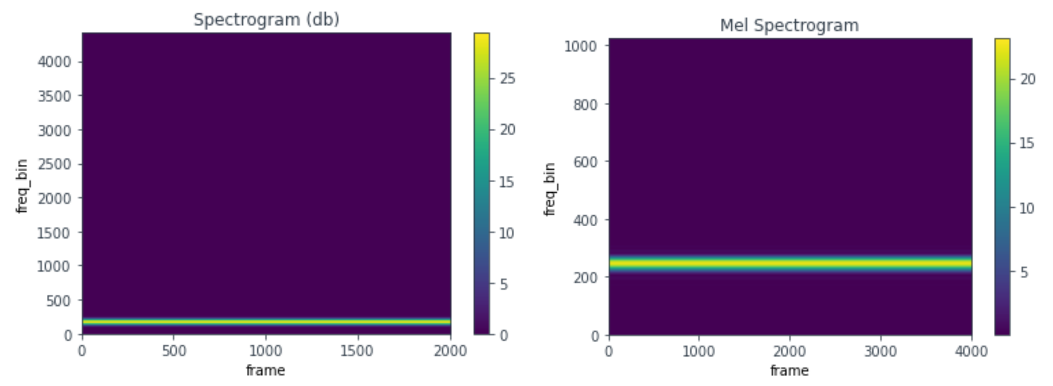
Using wolfram alpha to solve this: β=0.850854

Using this result, I set up the first layer using this decay value:

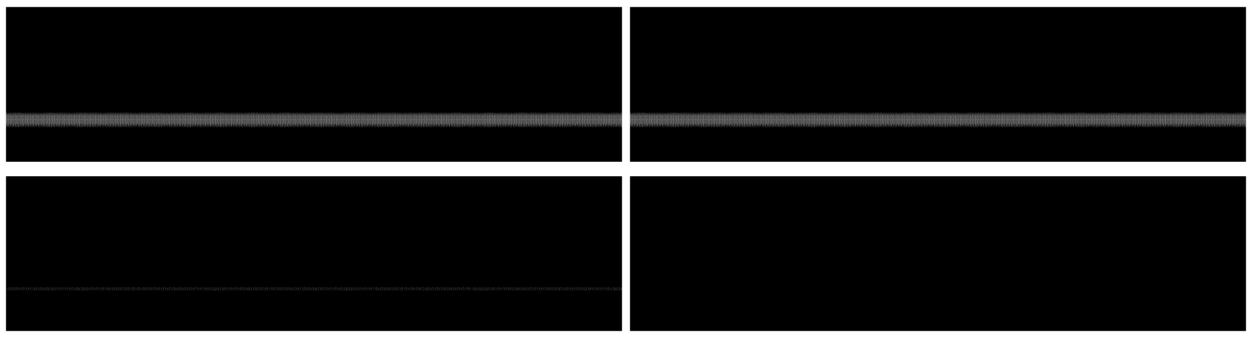
```
snn.Leaky(
            beta=0.850854,
            learn_beta=False,
            learn_threshold=False,
            threshold=1,
            reset_mechanism='zero',
            spike_grad=default_spike_grad,
            init_hidden=True)
```

On a layer that learns nothing but has a spike train that was engineered from the range of human hearing of a pure 800 Hz tone.

The first issue we see is that when we transform this pure tone into the Mel Frequency domain, and to a lesser extent, even in the frequency domain, the signal “bleeds” into the neighbouring buckets:

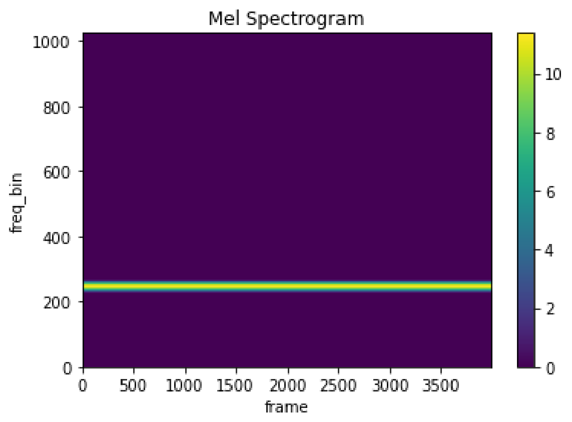


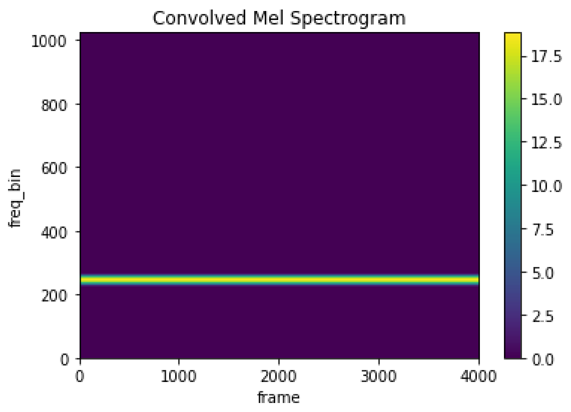
And as I examined the spike trains at amplitudes: 1, 0.1, 0.0004, 0.0001 I saw two things. The modelling for an appropriate Beta value was successful, and the bleeding of the signal into neighbouring buckets caused by spectral leakage shows up (as one would expect) in the temporal spike train of the model given the signal (the x axis represents time and the y axis represents signalling neurons, white on fire, black otherwise):



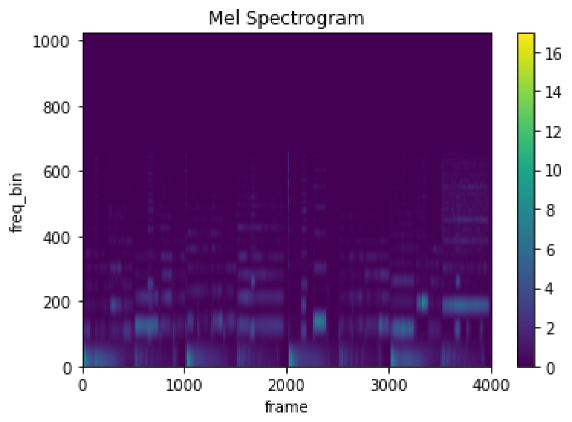
This actually is interesting in the context of the human cochlea because a similar physical phenomenon is likely to happen to neighbouring hairs in the basilar membrane. And for us to differentiate frequencies, it must work such that only the most excited frequencies (the hairs most pressed against the opposing side of the basilar membrane) are perceived (The Open University, 2018).

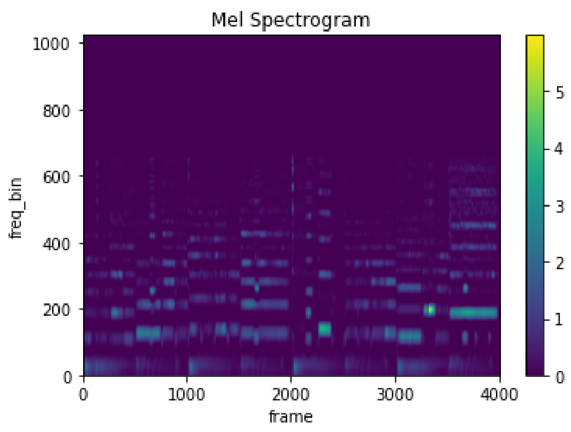
As a preprocessing step, there was a basic convolution that was helpful to magnify dominant signals with a 1-dimensional filter that is flanked on the left and right with negative numbers trigonometrically decaying in magnitude as they diverge from the centre. This was implemented as a single convolution matrix that gets applied after the mel spectrum conversion. Looking at the 800 kHz signal:



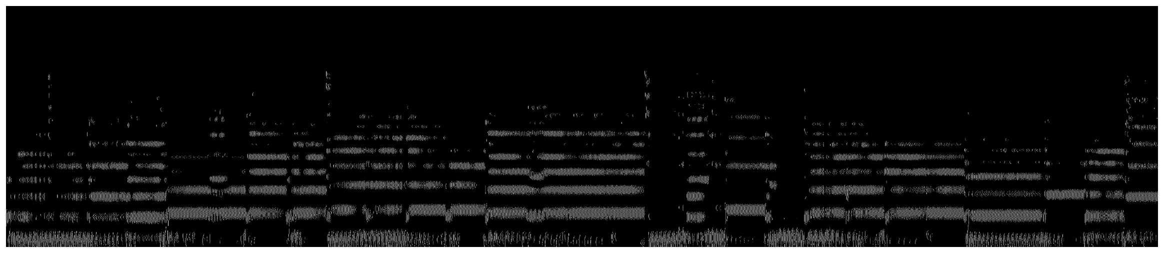


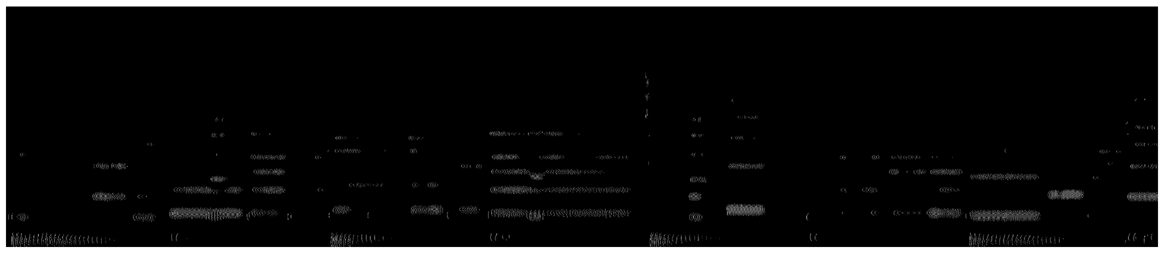
With the reduced image size it may be hard to see the very thin activation line on the convolved model. When viewing the real audio sample, the difference is a bit more apparent.





The convolved version gives us a similar spectrogram except with more contrast. Plotting the spike train this produces at full volume vs 1% volume:





Temporal Patterns in spike trains (TemporalPatterns.ipynb)

A single signalling neuron is insufficient for detecting patterns over time. However, unlike a traditional ANN neuron, a signalling neuron can “remember” things that happened in the past at least up to a spike. This makes them a viable building block for neural networks that can recognize patterns that unfold over time. My next task is to build on previous work to train an SNN to recognize a periodic pattern in a spike train arriving from a single input neuron. This will be important for the utility of training a network to differentiate between an acoustic flute with periodic fluctuations in breath vs a synthesised flute with a constant frequency band and amplitude.

Some ideas I’ve developed to no success so far:

Modelling a Fourier Transform as a moving average rather than an exact summation.

Convolving the spikes on a first layer into N SNN synaptic output neurons per input neuron. With each output having a different alpha beta so as to capture spikes over longer temporal patterns. This approach actually showed some success but it struggled to converge to an optimal solution so even after achieving 87% accuracy, it could quickly degrade again back to chance.

Provisional Conclusion

At this stage in my research, I have fallen a week behind by not yet successfully managing to model periodic temporal patterns coherently with SNNs yet. Even by my own schedule though, I would not have expected to have developed any final conclusions at this stage of my project. I did actually run a black box train and test cycle for noiseless audio data at the start and achieved a significant result prior to exploring and tuning the specific mechanics such that my approach would be equipped to handle noise. The specific goal and expectation that I have is that by modelling these additional challenges and exploring them, I can achieve at least equivalent accuracy robustly, even with noisy data.

References

Audacity (2022) *Audacity*, *Audacity ®*. Available at: https://www.audacityteam.org (Accessed: 30 July 2022).
Chadwick, J. (2022) *Google's DeepMind says it is close to achieving human-level AI*, *Mail Online*. Available at: https://www.dailymail.co.uk/sciencetech/article-10828641/Googles-DeepMind-says-close-achieving-human-level-artificial-intelligence.html (Accessed: 22 May 2022).
Chollet, F. (2018) *Deep learning with Python*. Shelter Island, New York: Manning Publications Co.
Collins, N. (2010) *Introduction to computer music*. Hoboken: John Wiley & Sons.
Cosyne 2022 Tutorial on Spiking Neural Networks - Part 1/2 (2022). Available at: https://www.youtube.com/watch?v=GTXTQ_sOxak (Accessed: 22 May 2022).
Dinchak, T. (2022) 'node-easymidi'. Available at: https://github.com/dinchak/node-easymidi (Accessed: 23 July 2022).
Drepper, U. (2007) 'What Every Programmer Should Know About Memory', p. 114.
endolith@gmail.com (2018) *Training neural network to implement discrete Fourier transform (DFT/FFT)*, *Gist*. Available at: https://gist.github.com/endolith/98863221204541bf017b6cae71cb0a89 (Accessed: 24 April 2022).
Engel, J. *et al.* (2017) 'Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders', *arXiv:1704.01279 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1704.01279 (Accessed: 23 April 2022).
Engel, J. *et al.* (2019) 'GANSYNTH: ADVERSARIAL NEURAL AUDIO SYNTHESIS', p. 17.
Engel, J. *et al.* (2020) 'DIFFERENTIABLE DIGITAL SIGNAL PROCESSING', p. 19.
Eshraghian, J.K. *et al.* (2022) 'Training Spiking Neural Networks Using Lessons From Deep Learning'. arXiv. Available at: http://arxiv.org/abs/2109.12894 (Accessed: 21 June 2022).
Fang, W. *et al.* (2020) *SpikingJelly*. Available at: https://github.com/fangwei123456/spikingjelly (Accessed: 21 June 2022).
Gemmeke, J.F. *et al.* (2017) 'Audio Set: An ontology and human-labeled dataset for audio events', in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA: IEEE, pp. 776–780. Available at: https://doi.org/10.1109/ICASSP.2017.7952261.
Guo, Y. *et al.* (2019) 'Unsupervised Learning on Resistive Memory Array Based Spiking Neural Networks', *Frontiers in Neuroscience*, 13, p. 812. Available at: https://doi.org/10.3389/fnins.2019.00812.
Gütig, R. and Sompolinsky, H. (2006) 'The tempotron: a neuron that learns spike timing–based decisions', *Nature Neuroscience*, 9(3), pp. 420–428. Available at: https://doi.org/10.1038/nn1643.
Hamdy, A., Vedula, P.K. and Konduru, M.V.J. (2019) 'Audio Separation and Isolation: A Deep Neural Network Approach', p. 6.
Harris, C.R. *et al.* (2020) 'Array programming with NumPy', *Nature*, 585(7825), pp. 357–362. Available at: https://doi.org/10.1038/s41586-020-2649-2.
Hershey, S. *et al.* (2017) 'CNN Architectures for Large-Scale Audio Classification', *arXiv:1609.09430 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1609.09430 (Accessed: 23 April 2022).
Hornik, K., Stinchcombe, M. and White, H. (1989) 'Multilayer feedforward networks are universal approximators', *Neural Networks*, 2(5), pp. 359–366. Available at: https://doi.org/10.1016/0893-6080(89)90020-8.
Hu, D. *et al.* (2020) 'Curriculum Audiovisual Learning'. arXiv. Available at:

http://arxiv.org/abs/2001.09414 (Accessed: 21 June 2022).
Huang, C.-Z.A. *et al.* (2018) ‘Music Transformer’, *arXiv:1809.04281 [cs, eess, stat]* [Preprint]. Available at: http://arxiv.org/abs/1809.04281 (Accessed: 23 April 2022).
Kamata, H., Mukuta, Y. and Harada, T. (2021) ‘Fully Spiking Variational Autoencoder’, *arXiv:2110.00375 [cs]* [Preprint]. Available at: http://arxiv.org/abs/2110.00375 (Accessed: 23 April 2022).
Koretzky, A. (2019) *Audio AI: isolating instruments from stereo music using Convolutional Neural Networks*, *Medium*. Available at: https://towardsdatascience.com/audio-ai-isolating-instruments-from-stereo-music-using-convolutional-neural-networks-584ababf69de (Accessed: 23 April 2022).
López-Randulfe, J. *et al.* (2021) ‘Spiking Neural Network for Fourier Transform and Object Detection for Automotive Radar’, *Frontiers in Neurorobotics*, 15, p. 688344. Available at: https://doi.org/10.3389/fnbot.2021.688344.
McDermott, J. (2020) *Successes and Failures of Neural Network Models of Hearing | The Center for Brains, Minds & Machines*. Available at: https://cbmm.mit.edu/video/successes-and-failures-neural-network-models-hearing (Accessed: 20 June 2022).
McFee, B. *et al.* (2022) ‘librosa/librosa: 0.9.2’. Zenodo. Available at: https://doi.org/10.5281/ZENODO.6759664.
Mesaros, A. *et al.* (2010) ‘Acoustic event detection in real life recordings’, p. 5.
Mobin, S., Cheung, B. and Olshausen, B. (2018) ‘Generalization Challenges for Neural Architectures in Audio Source Separation’, *arXiv:1803.08629 [cs, eess]* [Preprint]. Available at: http://arxiv.org/abs/1803.08629 (Accessed: 23 April 2022).
Narayanaswamy, V. *et al.* (2020) ‘Unsupervised Audio Source Separation using Generative Priors’, *arXiv:2005.13769 [cs, eess, stat]* [Preprint]. Available at: http://arxiv.org/abs/2005.13769 (Accessed: 23 April 2022).
Neftci, E.O., Mostafa, H. and Zenke, F. (2019) ‘Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks’, *IEEE Signal Processing Magazine*, 36(6), pp. 51–63. Available at: https://doi.org/10.1109/MSP.2019.2931595.
van den Oord, A. *et al.* (2016) ‘WaveNet: A Generative Model for Raw Audio’, *arXiv:1609.03499 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1609.03499 (Accessed: 2 May 2022).
OpenAI GPT-3: Everything You Need To Know (2021) *Springboard Blog*. Available at: https://www.springboard.com/blog/data-science/machine-learning-gpt-3-open-ai/ (Accessed: 22 May 2022).
Payeur, A. *et al.* (2020) *Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits*. preprint. Neuroscience. Available at: https://doi.org/10.1101/2020.03.30.015511.
Roberts, A. *et al.* (2019) ‘A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music’, *arXiv:1803.05428 [cs, eess, stat]* [Preprint]. Available at: http://arxiv.org/abs/1803.05428 (Accessed: 8 May 2022).
The Open University (2018) *Hearing*, *Hearing*. Available at: https://www.open.edu/openlearn/science-maths-technology/biology/hearing/science-maths-technology/biology/hearing (Accessed: 5 June 2022).
Toews, R. (2020) *Deep Learning’s Carbon Emissions Problem*, *Forbes*. Available at: https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/ (Accessed: 22 May 2022).
Vreeken, J. (2003) ‘Spiking neural networks, an introduction’, p. 5.
Wu, J. *et al.* (2018) ‘A Spiking Neural Network Framework for Robust Sound Classification’, *Frontiers in Neuroscience*, 12, p. 836. Available at: https://doi.org/10.3389/fnins.2018.00836.
YouTube (2022) *YouTube for Press*, *blog.youtube*. Available at: https://blog.youtube/press/

(Accessed: 2 May 2022).
Yu, C. *et al.* (2018) ‘Multi-level Attention Model for Weakly Supervised Audio Classification’, *arXiv:1803.02353 [cs, eess]* [Preprint]. Available at: http://arxiv.org/abs/1803.02353 (Accessed: 23 April 2022).
Zheng, H. *et al.* (2020) ‘Going Deeper With Directly-Trained Larger Spiking Neural Networks’, *arXiv:2011.05280 [cs]* [Preprint]. Available at: http://arxiv.org/abs/2011.05280 (Accessed: 24 April 2022).
Zhu, F. *et al.* (2018) ‘Architecture of the Mouse Brain Synaptome’, *Neuron*, 99(4), pp. 781-799.e10. Available at: https://doi.org/10.1016/j.neuron.2018.07.007.