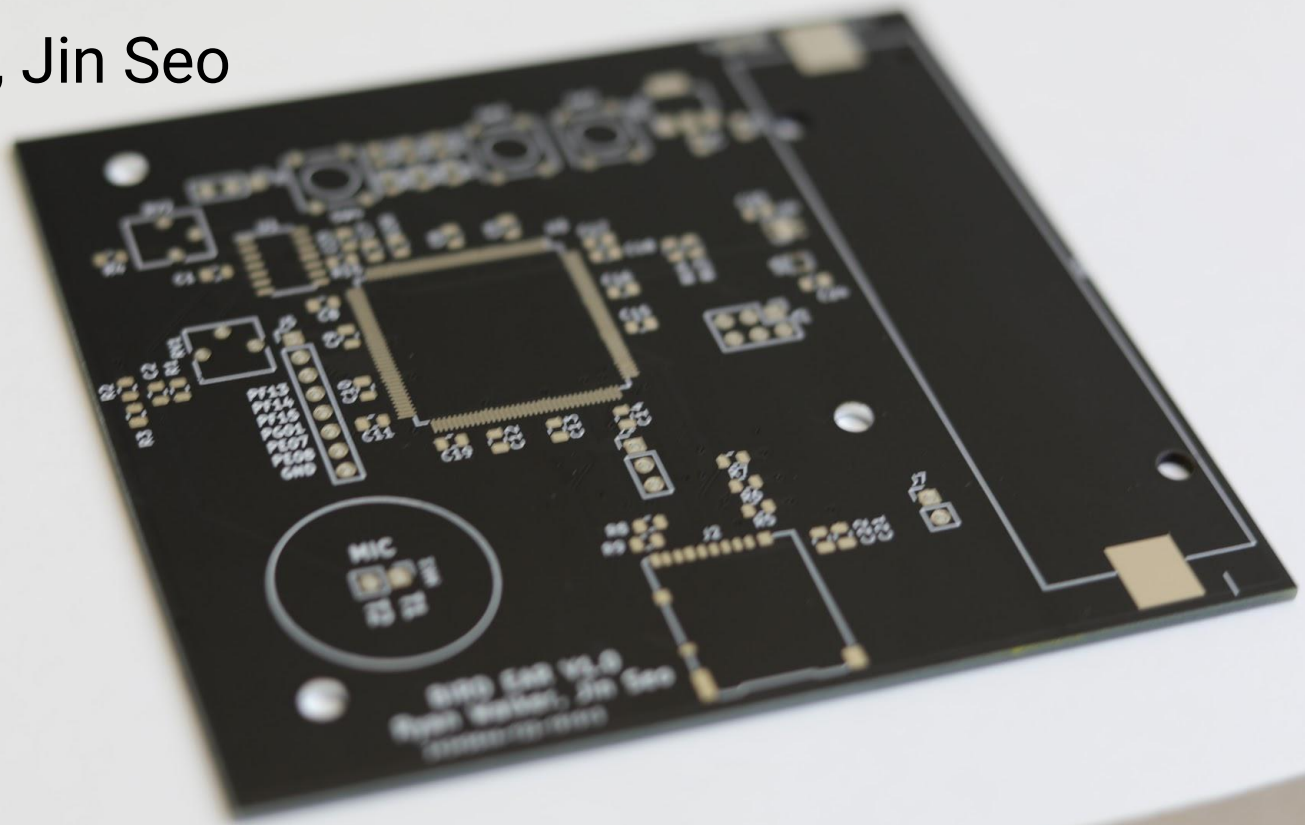
BirdEar

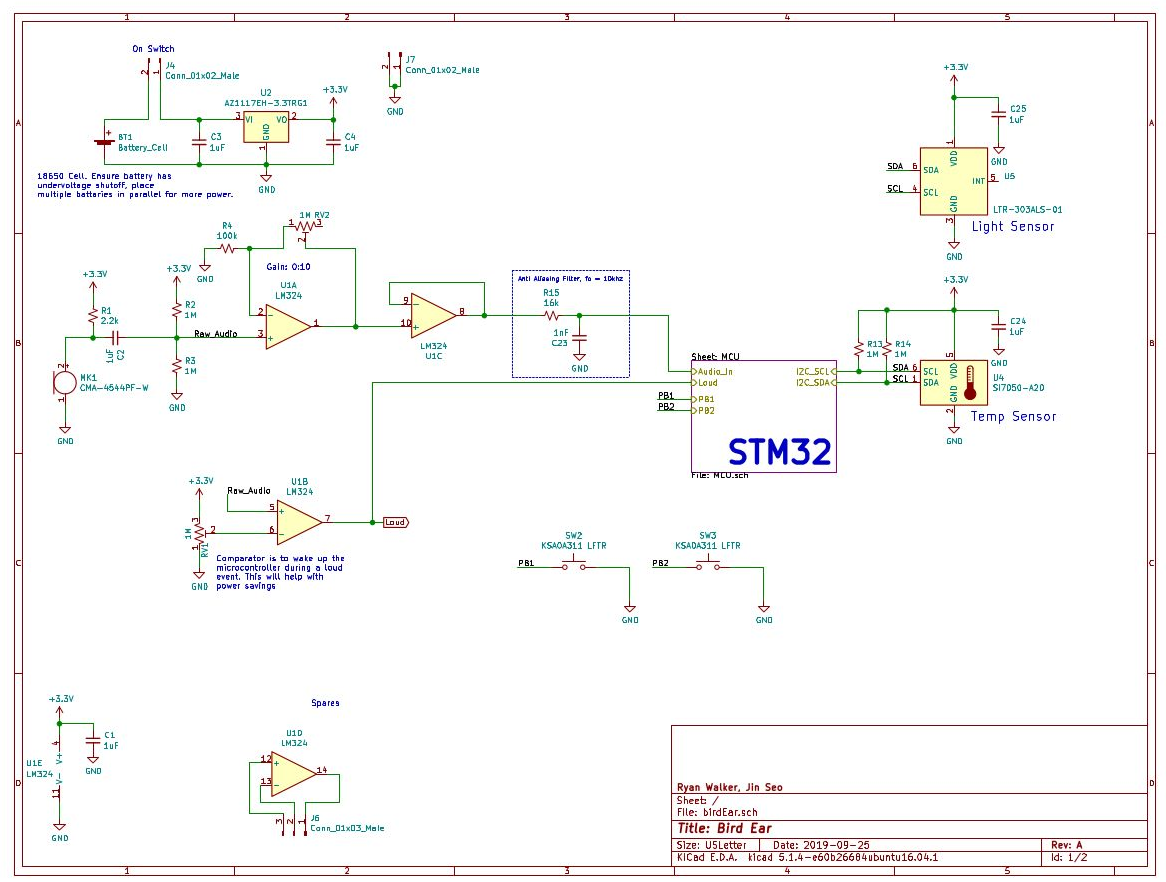
Ryan Walker, Jin Seo



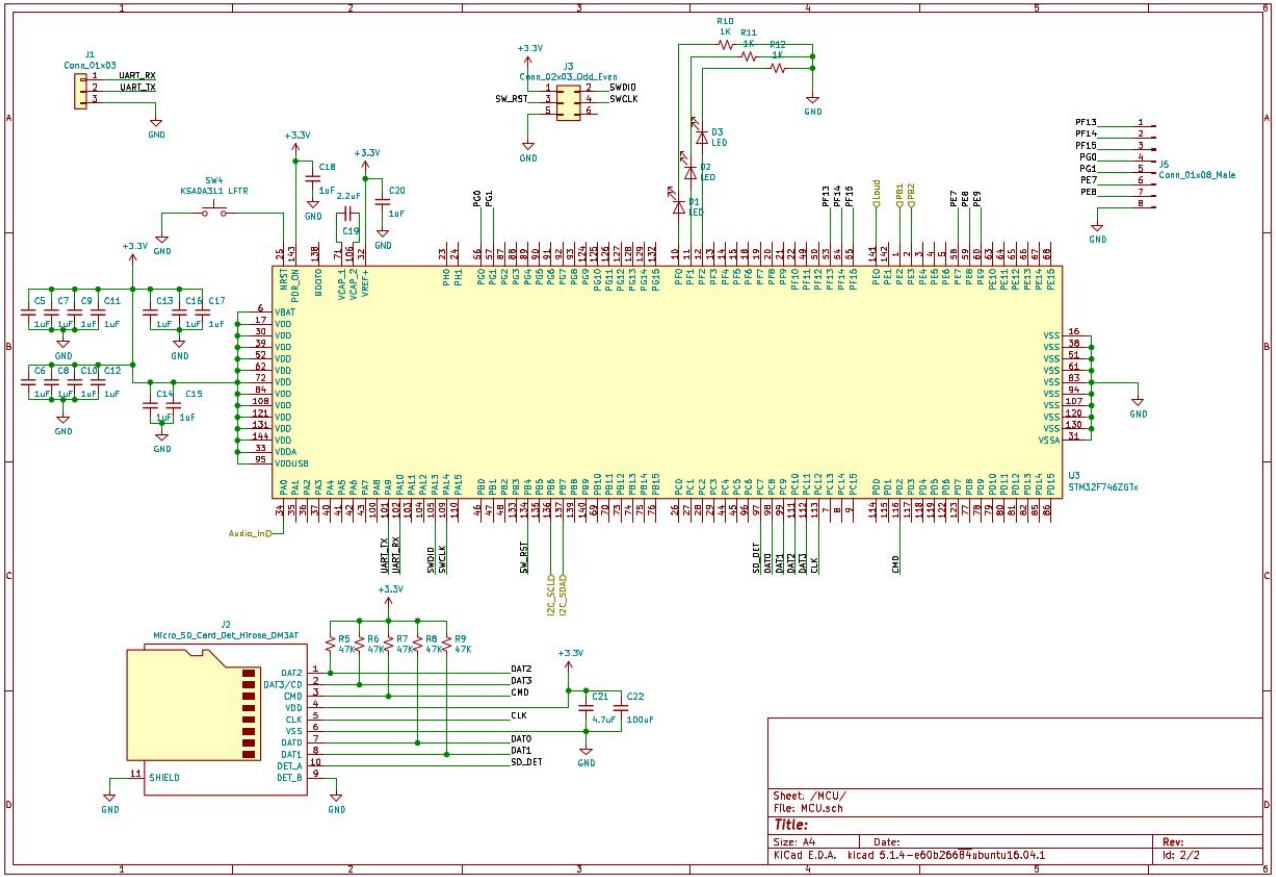
BirdEar

- Aiming to understand bird density in a given area through acoustic analysis
    - This will help biologist understand bird migration over time
- Design hardware that can be deployed and persistently exist in the field
- Develop algorithms to accurately distinguish between different species of birds using exclusively acoustic data
    - Then log this data to an SD card with a timestamp

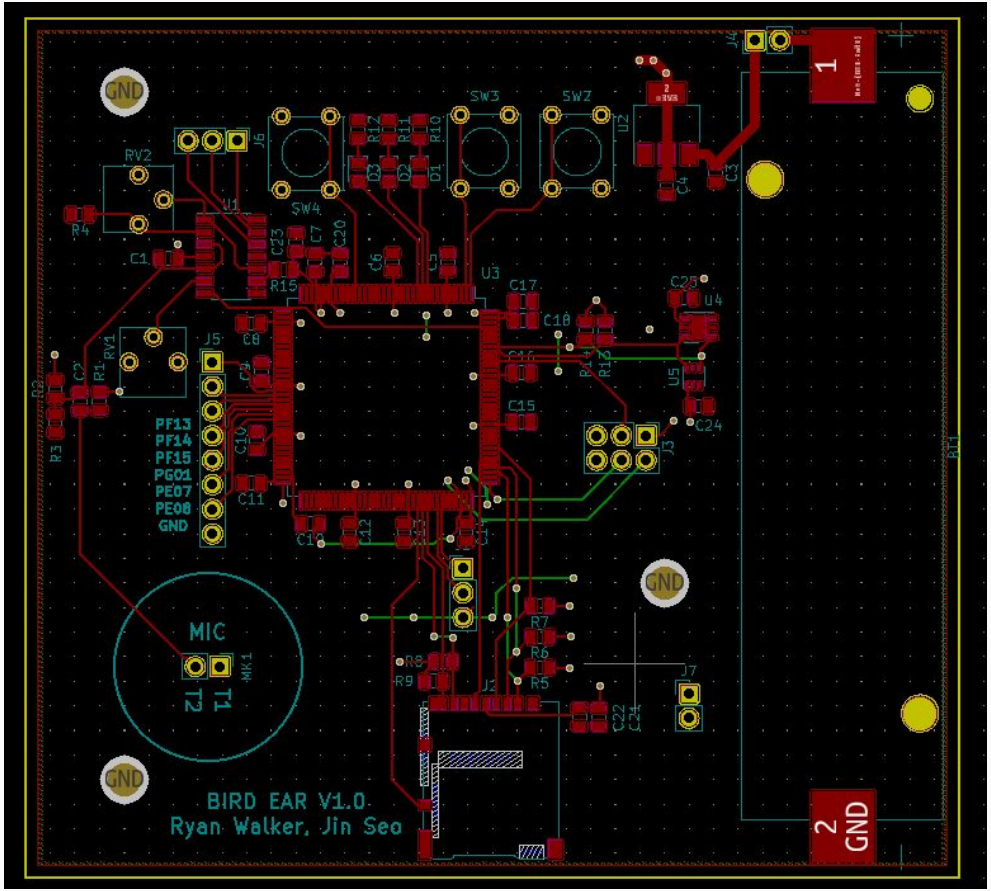
Hardware - Design



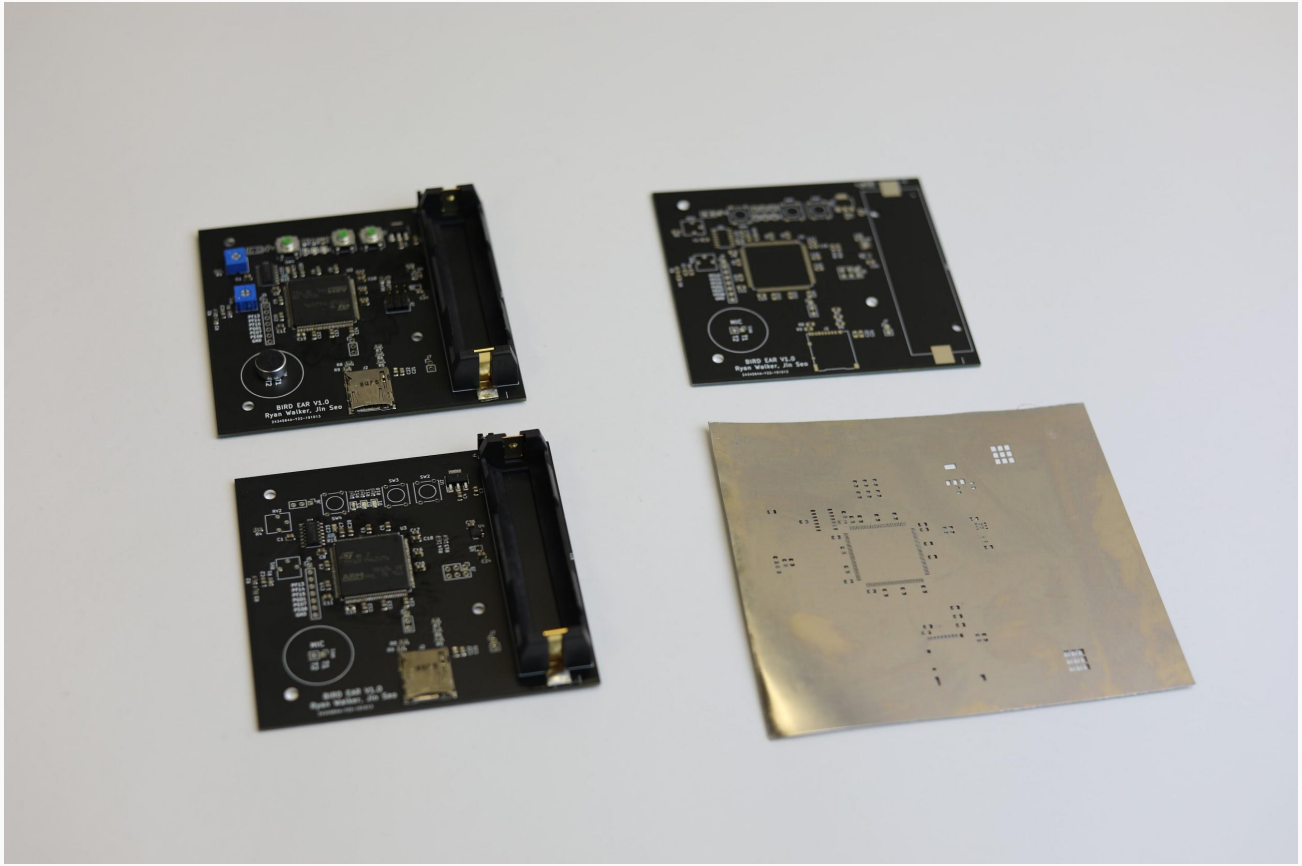
Hardware - Design



Hardware - Layout



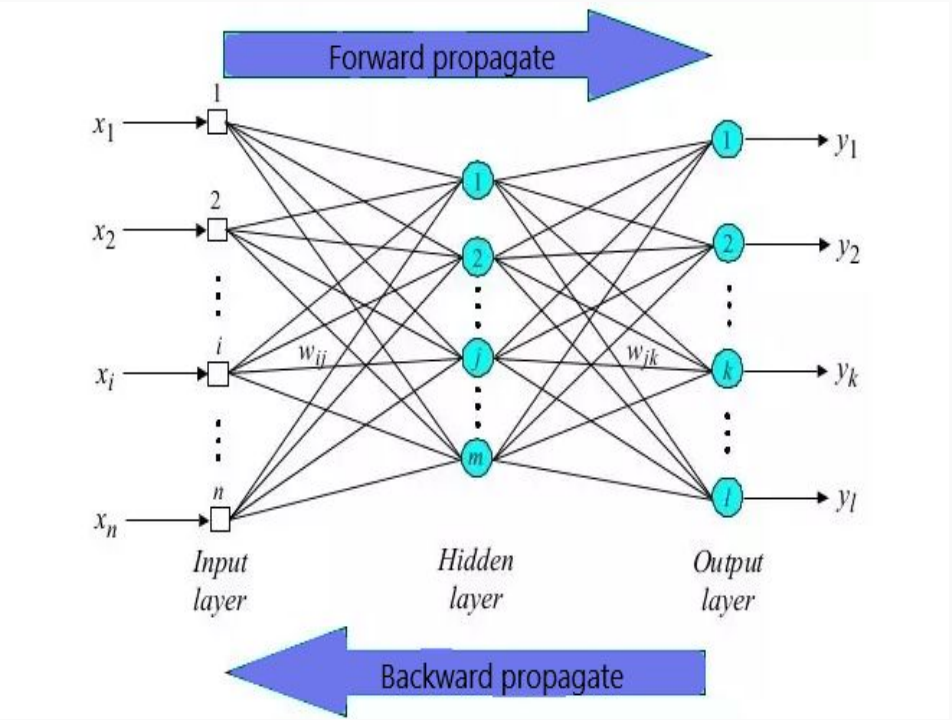
Hardware


BIRD EAR V1.0
Ryan Walker, Jin Seo
MIC
BIRD EAR V1.0
Ryan Walker, Jin Seo
MIC
BIRD EAR V1.0
Ryan Walker, Jin Seo

Neural Network (NN)

- Artificial **Neural Networks** work on the basis of the structure and functions of a human brain. A human brain consists of neurons that process and transmit information between themselves

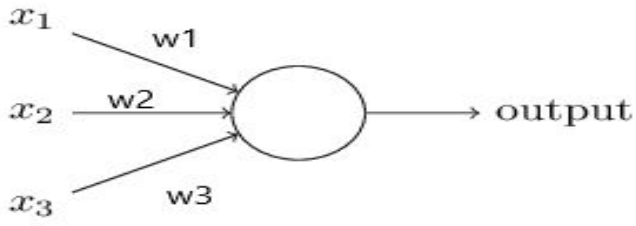
- **Neural network** is a system of equations that processes existing classified data with the goal of obtaining learnable parameters, which are called weights.



Perceptron - Early model of NN

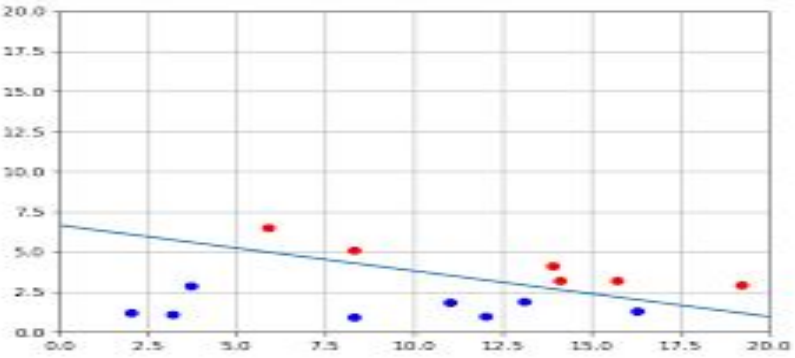
- Used perceptrons to model of decision-making.
- The larger value of w indicates that matters a lot to make a decision

$$Output = \begin{cases} 0, & if\ w \cdot x + b \leq 0 \\ 1, & if\ w \cdot x + b > 0 \end{cases}$$



xj is inputs and wj is weights, b is bias

- **Perceptrons act as a linear decision; however, many reality functions are NOT linear.**



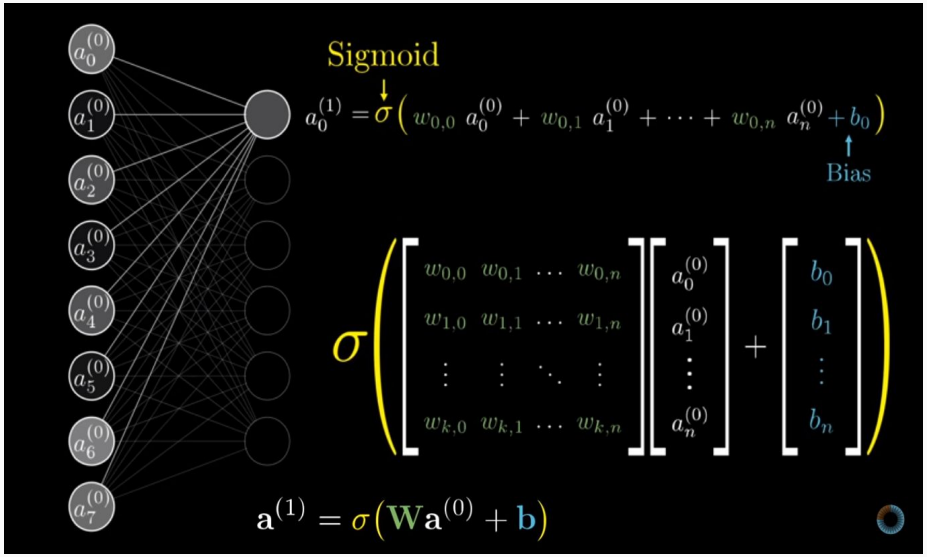
Sigmoid neuron

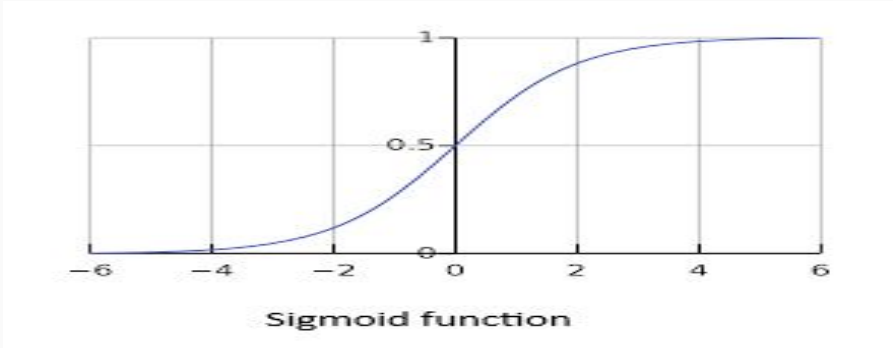
- Sigmoid neuron has weights for each input w1,w2, …., and overall bias b. It's **σ(w·x+b)** where is called the **sigmoid function**

$$\sigma(z) = \frac{1}{1+e^{-z}} \ldots (3)$$

- **The output of a sigmoid neuron with inputs** x1, x2, …, **weights** w1,w2,…, **and bias b**

$$\frac{1}{1+\exp\left(-\sum_j w_j x_j - b\right)} \ldots (4) \text{ where } z = w \cdot x + b$$





Learning with Gradient descent

- $C = \frac{1}{n}\sum_x C_x$, C *is an average over the cost function* C_x

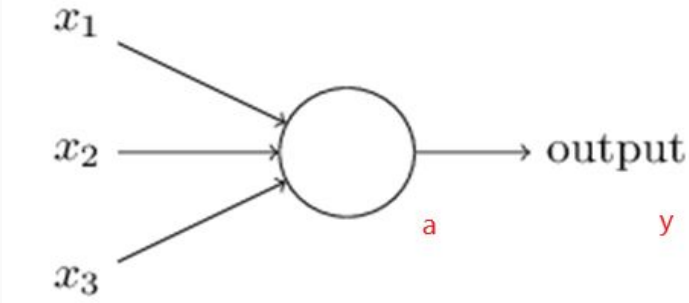
  The quadratic cost function $C = \frac{1}{2n}\left|\left|y - a^l\right|\right|^2$

  Let's

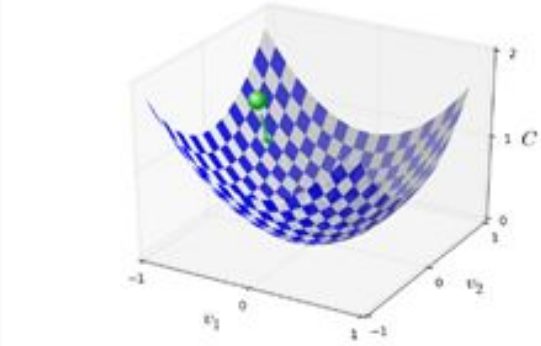
  $v = y(x) - a$ *and* $||v||$ *denotes the usual length for vector* v,

  $y(x)$ *is expected output,* a *is the output of neuron*

  $a^l = \sigma(Wa^{l-1} + b^l)$



- We want to find a set of weights and biases which make the cost as small as possible, known as **gradient descent.**



minimize cost function C(v), where v=v1, v2.

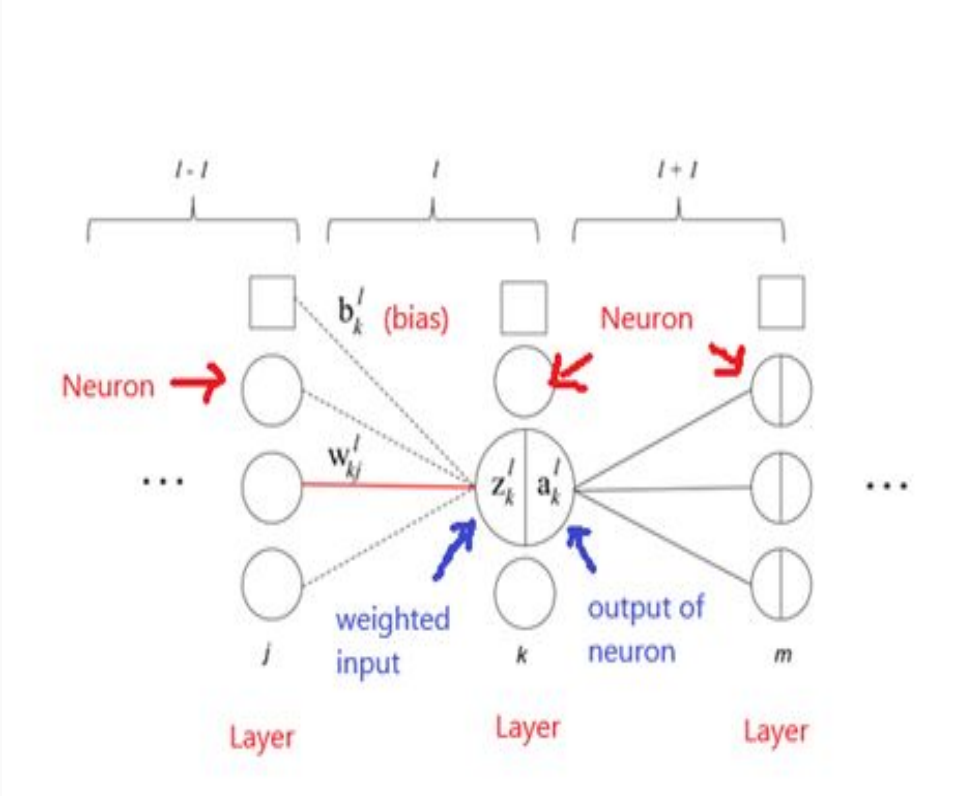
Learning with Gradient descent & Forward propagation

- $$a' = \sigma(wa + b) \ldots (22)$$

$$z_k^l = \sum_j w_{kj}^l a^{l-1} + b_k^l,$$

Where **a'** is the vector of activations of neuron in the third layer, **a** is the vector of activations of second layer of neurons. **z** is weighted input.

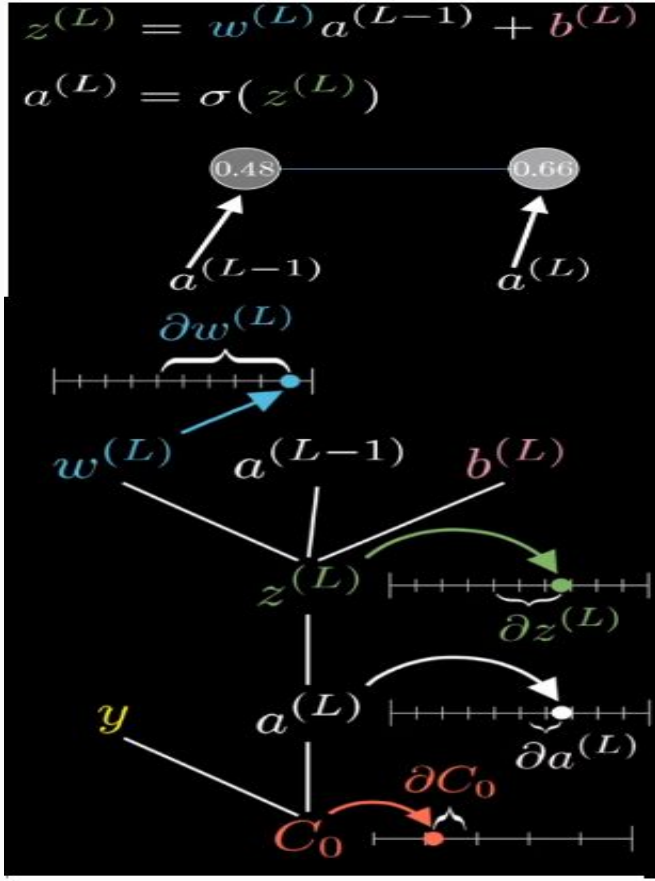
- Unfortunately, when the number of training inputs is very large this can take a long time and learning thus occurs slowly. So, **stochastic gradient descent** can be used to speed up learning. The idea is to estimate the gradient C by computing Cx for a small sample of randomly chosen training inputs.



Backward propagation

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Desired output

$$C_0(\dots) = (a^{(L)} - y)^2$$

z is weighted input
a is output of neuron
C is cost function
y is desired output

Chain rule

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

Backward propagation

- **Partial derivative & Chain rule**

$$z^{[l]} = w^{[l]}.a^{[l-1]} + b^{[l]}$$

$$\frac{\partial z^{[l]}}{\partial w^{[l]}} = \frac{\partial}{\partial w^{[l]}}(w^{[l]}.a^{[l-1]} + b^{[l]})$$

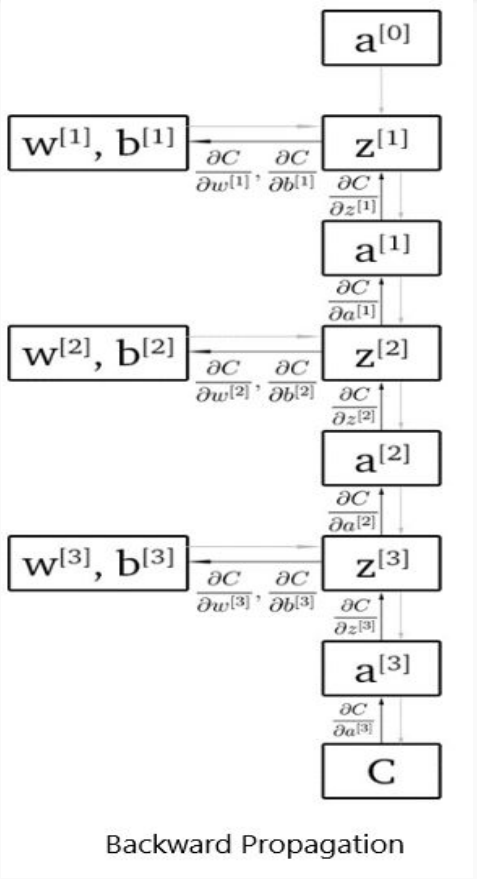
$$= a^{[l-1]}$$

$$\therefore \boxed{\frac{\partial z^{[l]}}{\partial w^{[l]}} = a^{[l-1]}}$$

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_m \frac{\partial C}{\partial z^l_m}\frac{\partial z^l_m}{\partial w^l_{jk}}, \quad \delta^l_j = \frac{\partial C}{\partial z^l_j}$$

$$\therefore \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j,$$

where δ^l_j is an error of j neuron in l layer



Backward Propagation

Cross-entropy Cost function

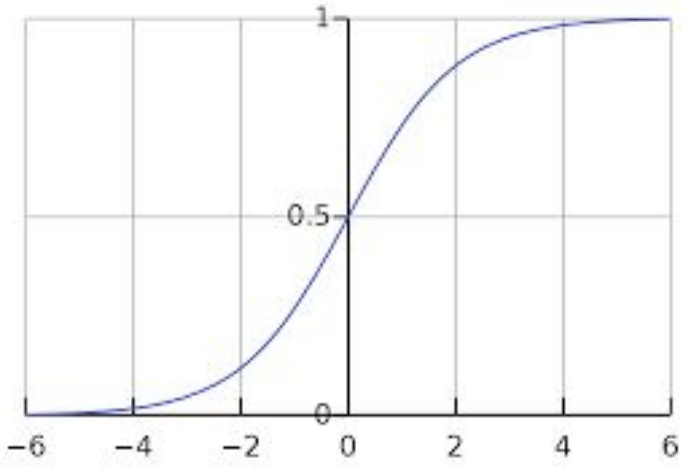
- As for the Sigmoid function, when the neuron's output is close to 1, the curve gets very flat, so $\sigma'(z)$ gets very small (slow down).

$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L} = a^{L-1}\sigma'(z^L)2(a^L - y)$$

- **Cross-entropy avoids the problem of learning slowdown by vanishing the term $\sigma'(z_j^L)$**

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)] \dots (57), \text{where } a = \sigma(z)$$

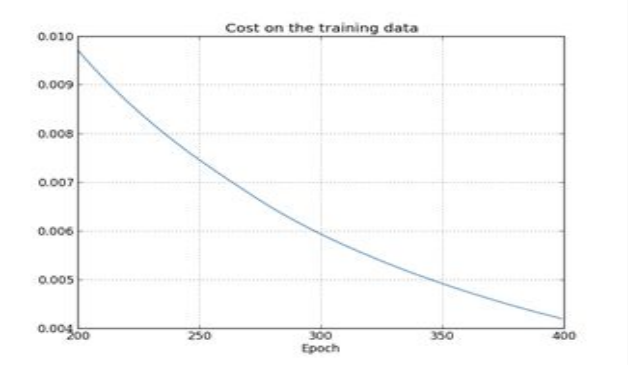
$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y) \dots (61)$$



Sigmoid function

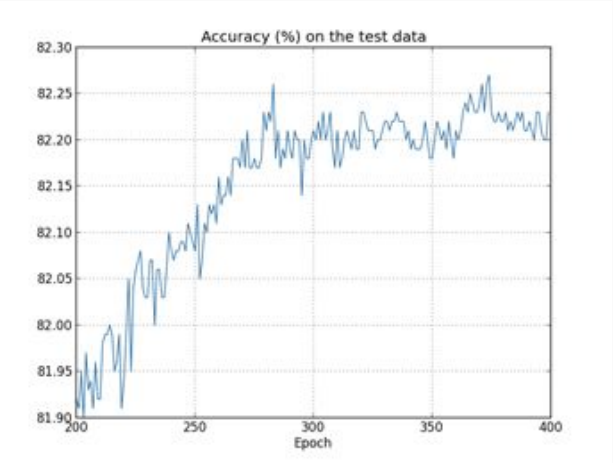
Training & Validation & Early stopping

- In training process, separate **training and validating data** so as to evaluate NN model
- **Over fitting (Over training)** is a situation where the **cost of training data decreases** but **the accuracy of validating** data gets **saturated**



- Once the classification accuracy on the validation data has saturated**, we stop training (Early stopping)** to **avoid overfitting.** Then save the **hyper-parameters (# of epochs, learning rate,...).**

  Repeating training by modifying the NN model and parameters until getting good accuracy

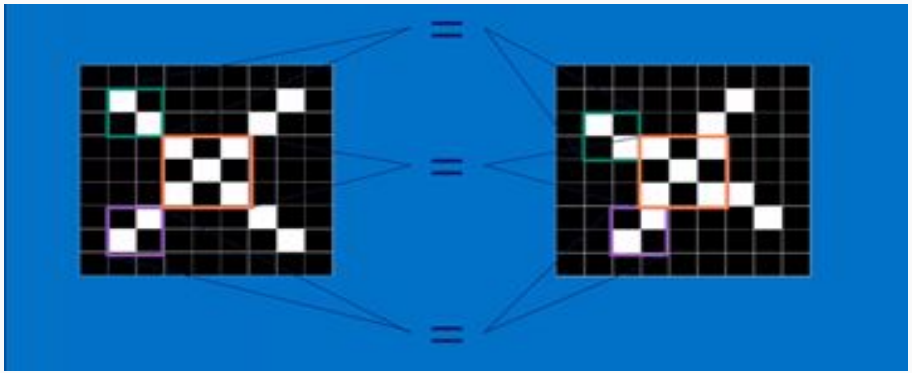


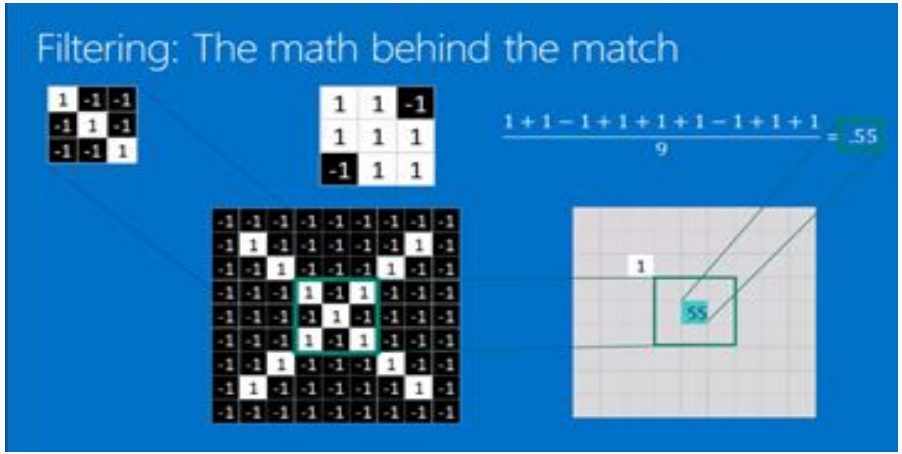
Other NN model - Convolutional NN

- Better NN model for image and speech recognition
- CNN is to match parts of the image(data)-filtering- rather than whole thing by breaking down smaller parts.
- $$\sigma(b + \sum_{l=0}^{n}\sum_{m=0}^{n} w_{l,m} a_{j+l,k+m}) \quad ...(125)$$

  Filtering steps:

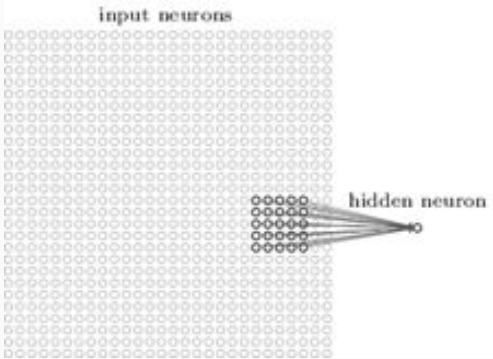
  1. Line up the feature and map
  2. Multiply each element of map by corresponding feature element
  3. Add them up
  4. Divide by the total number of elements in the feature

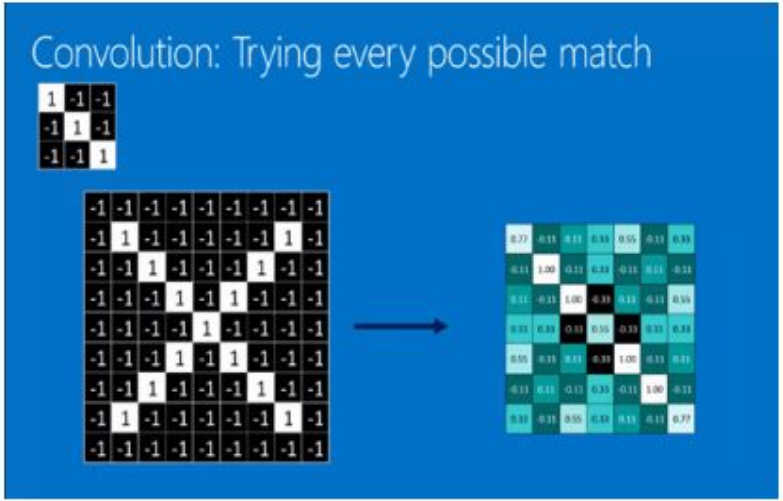




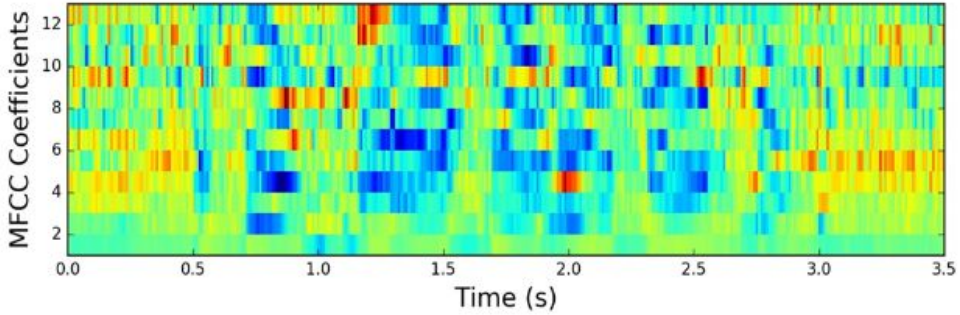
Convolutional NN (CNN)

- A big advantage of sharing weights and biases is that it greatly **reduces the number of parameters** involved in a convolutional network.





End of convultion



Spectrogram of MFCC

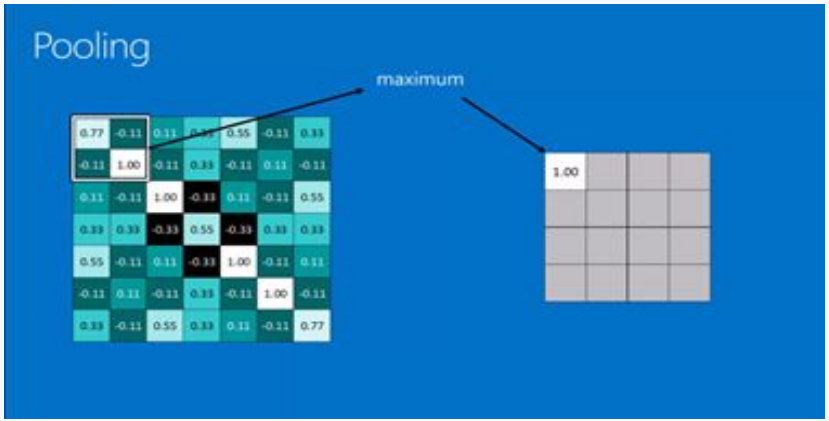
Max-pooling

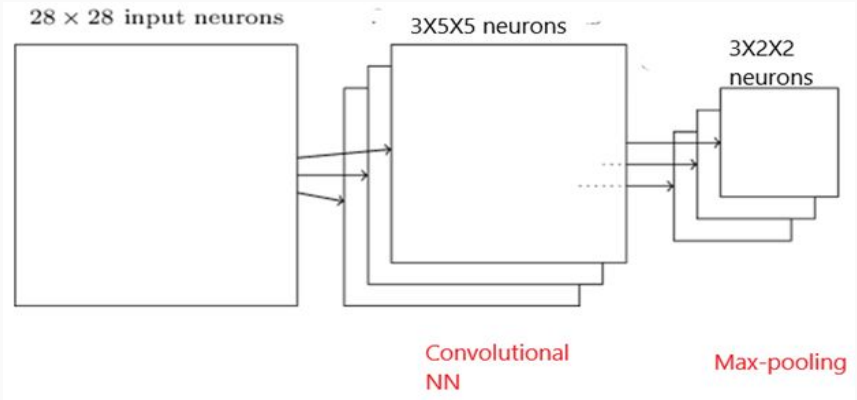
- In **max pooling**, a pooling unit simply outputs the **maximum activation** in the N×N input region

  Pooling steps :

  1. Pick window size(2 or 3)
  2. Pick stride size (usually 2)
  3. Walk the window across the filtered outputs (CNN)
  4. From each window, take the maximum value

- A big benefit is that this helps **reduce the number of parameters** needed in later layers.





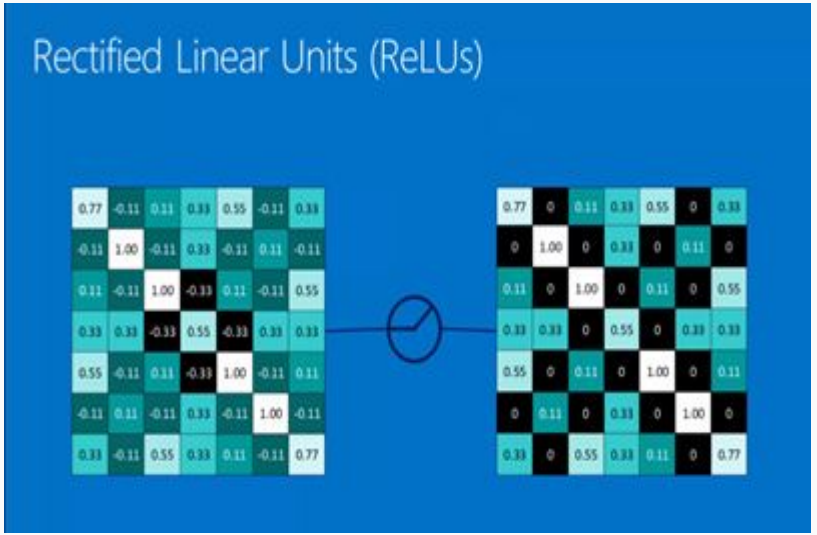
Other activation function - Rectified Linear Units (ReLU)

- **Normalizing** by tweaking each of values just a bit

  ex) change every negative values to zero

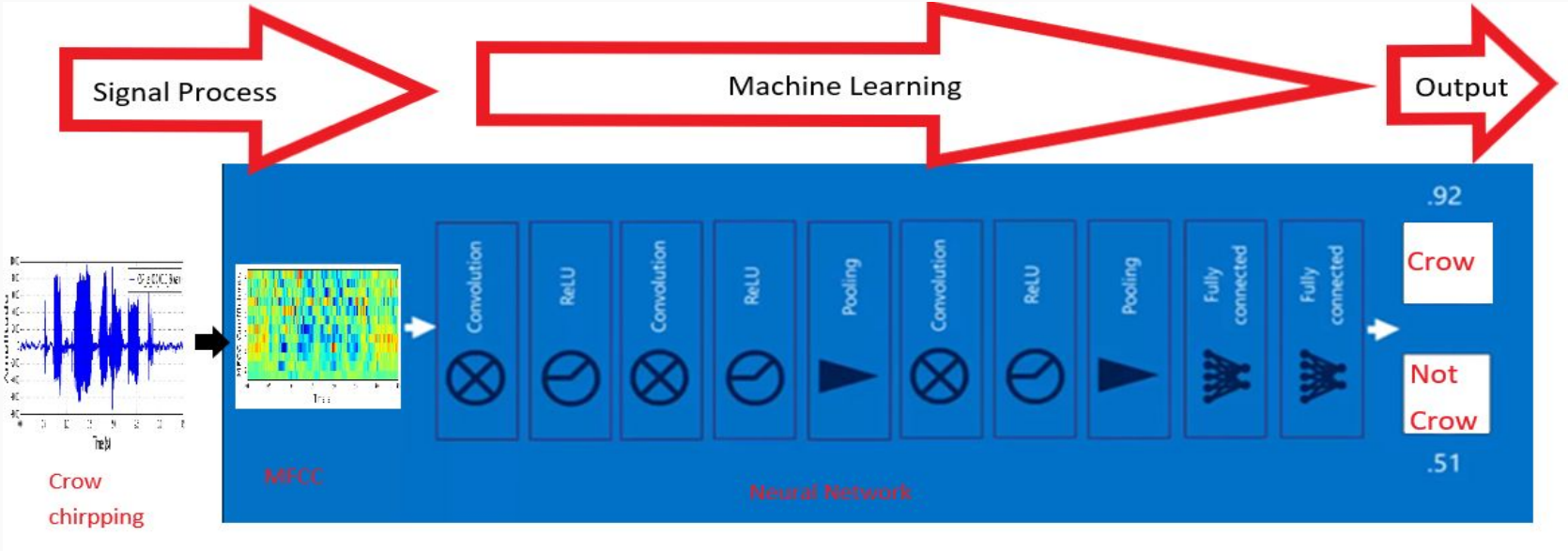
- Found that networks based on rectified linear units **consistently outperformed** than networks based on sigmoid activation functions.

  Empirically better (Not mathematically proved)



Put all together

- For better result of NN
  We can use the output from a layer to the input of next layer (Multi-layers).



Fully-Connected layer means **NO Convolutional NN**

Neural Network Implementation

- We have harvested thousands of sound sample from Crows and Sandpipers
- This data was then moved into fourier space (STFT or FFT)
- We then implement mel-spaced frequency array(MFCC) that feeds into NN
- We achieved ~93% accuracy on our validation set (Python)
- This model was then converted to a “TF lite” flatpack (For MCU)

Next Steps

- Increase the number output neurons to 2, so we can classify between crows and sandpipers.
- Add another three types of birds to the dataset.
- Research "VGG like feature extraction".
- Implement microcontroller firmware for audio sampling.
- Implement neural network on the microcontroller.
- Deploy the device in the field.

Thanks!