Institute Technical Summer Project
Indian Institute of Technology, Bombay
Powai, Mumbai - 400076, INDIA

Website: https://github.com/bvcxz1/ITSP

Formulae - itsp.py

Writing in Thin Air

Signal Processing:

- **Transformation Matrix** A transformation matrix is used to reduce the number of dimensions of the acceleration input from 3 to 2 by eliminating the null space in the signal input.

  The first matrix changes the frame of reference from the body frame of the sensor to an inertial frame

$$T_1 \rightarrow \begin{bmatrix} x_1 \\ y_1 \\ z_0 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{1}$$

  The second matrix eliminates the null space

$$T_2 \rightarrow \begin{bmatrix} x_2 \\ y_1 \\ 0 \end{bmatrix} = \begin{bmatrix} cos\phi & 0 & sin\phi \\ 0 & 1 & 0 \\ -sin\phi & 0 & cos\phi \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_0 \end{bmatrix} \tag{2}$$

  where $[x_2\ y_1\ 0]^t$ is called *sensor_output_red*

- **Moving Average Filter:** A filter used to remove high frequency noise from raw data

$$filtered\_output\_ma = \frac{1}{n} \sum_{i=0}^{N-n+1} \sum_{j=i}^{i+n} sensor\_output\_red_j \tag{3}$$

  where n is a variable over which the number of recent readings are averaged

- **High Pass Filter:** A filter used to remove gravitational acceleration from raw data

$$filtered\_output\_hp = filtered\_output\_ma - g \tag{4}$$

  where *g* is the acceleration due to gravity

Neural Network:

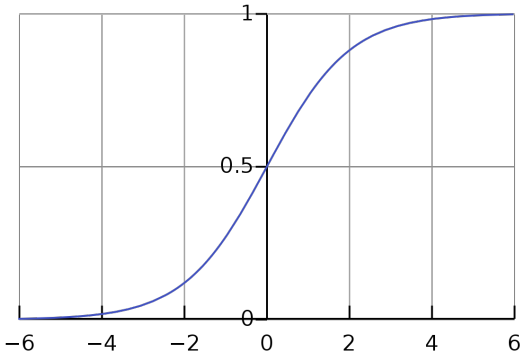
- **Sigmoid Neurons, *also called logistic neurons***

  The sigmoid neuron has inputs, $x_1, x_2, \ldots$.which can take on any values between 0 and 1. So, for instance, $0.4238\ldots$.is a valid input for a sigmoid neuron. The sigmoid neuron has weights for each input, $w_1, w_2, \ldots$,and an overall bias, b. The output of $\sigma(w \cdot x + b)$ will be between 0 and 1, and is called the *sigmoid or logistic function*.

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}. \tag{5}$$

$$\sigma(z) \equiv \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{6}$$

The shape of the sigmoid function when plotted:



The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias will produce a small change $\Delta output$ in the output from the neuron. In fact, calculus tells us that $\Delta output$ is well approximated by:

$$\Delta \text{output} \approx \sum_j \frac{\partial \text{ output}}{\partial w_j} \Delta w_j + \frac{\partial \text{ output}}{\partial b} \Delta b, \tag{7}$$

where the sum is over all the weights, w_j, and $\delta output/\delta w_j$ and $\delta output/\delta b$ denote partial derivatives of the output with respect to w_j and b, respectively.

- **Cost Function, *also called loss or objective function*** To find weights and biases so that the output from the network approximates $y(x)$ for all training inputs x:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{8}$$

Here, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum is over all training inputs, x and C is called the quadratic cost function; it's also sometimes known as the *mean squared error* or just *MSE*. The aim of a training algorithm will be to minimize the cost function and will be done using an algorithm known as gradient descent.

- **Learning Rate** For small changes

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{9}$$

If we choose

$$\Delta v = -\eta \nabla C, v \to v' = v - \eta \nabla C. \tag{10}$$

where η is a small, positive parameter known as the learning rate then

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \tag{11}$$

and this guarantees that $\Delta C \leq 0$, i.e., C will always decrease. To make gradient descent work correctly, $\eta$is chosen to be small enough that Equation (9) is a good approximation and $\Delta C \leq 0$ holds true. At the same time, η can't be too small and the gradient descent algorithm will work very slowly.

- **Applying Gradient Descent** The idea is to use gradient descent to find the weights w_k and biases b_l which minimize the cost in Equation (8). Writing out the gradient descent update rule in terms of components:

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{12}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{13}$$

  The code implements a minimised version of Gradient Descent called the Stochastic Gradient Descent where the average value is not calculated for all the values but only for a few randomly chosen values.

- **Mini Batch Size** The factor by which the number of elements is reduced to apply Stochastic Gradient Descent

- **Epoch** The point at which all training inputs have been exhausted