# Intro to AI and ML

Principles behind the toy-car ML Engine

Soham Kulkarni
EE19BTECH11053
IIT HYDERABAD

## ***Overview***

**Data Handling(collection and dataset creation)

**Training the model

**Speech Recognition and Classification(Labeling)

## Data Handling- Zero Padding

Step 1: Recording 80 voice samples for each of the 5 commands.

Step 2: Zero Padding- Adding 0s in the voice samples in the beginning and the end so as to create new data in huge quantity for training.

In our case, 80 voice samples of each voice file are created, which means for each command;

80*250=20,000 samples conditioned.

```
new_data = np.empty([25000,]) #Array creation for new files
```

```
p = 25000-x
for y in range(1 ,p):   #Defining p and y
```

Adding empty elements in the start:

```
for i in range(0,y-1):      new_data[i] =y1[i]
```

Middle data indexed between (y) and (25000-x+y-1) is preserved:

```
for i in range(y,25000-x+y-1):
new_data[i] =data[i-y]
```

Adding empty elements in the end:

```
for i in range(25000-y , 24999):    new_data[i] = y1[i]
```

## Data Handling-Signal Conditioning

**Here, we convert our generated file into MFCC format.

**Mel-frequency cepstral coefficients** (**MFCCs**) are coefficients that collectively make up an MFC.

The **mel-frequency cepstrum** (**MFC**) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

**Basically, this is done for feature extraction from the voice samples.

**We extract 39 features for each timestep out of a total of 49 timesteps.

from python_speech_features import mfcc #Installing MFCC library

## Data handling-Making Datasets and Interleaving data

**Here, we link the input and the output in a single array for the generation of the training and the testing datasets.

**As for each command,  20,000 samples were conditioned in the Zero Padding step earlier, 80% of these samples, i.e. 16,000 samples are used as training sets.

**Now, data is interleaved in file format.  (**Interleaving** of data refers to the interspersing of fields or channels of different meaning sequentially in memory, in processor registers, or in file formats.)

**For example, for our given 49 timesteps, 39 features define each timestep, we now combine data of each time step corresponding to each feature and store it in a new array. It is similar to intermixing of data.

## Training the model: Initialization

**Weights are initialized usually randomly to reach the ideal weight matrix but this kind of initialization is prone to vanishing or exploding gradient problems.

**Tanh activation function to make our neural network initialize better weights:**

1. Generate random sample of weights from a Gaussian distribution having mean 0 and a standard deviation of 1.
2. Multiply that sample with the square root of (1/(ni+no)). Where ni is number of input units, no is the number of output units for that layer respectively.

## Training the model: Initialization

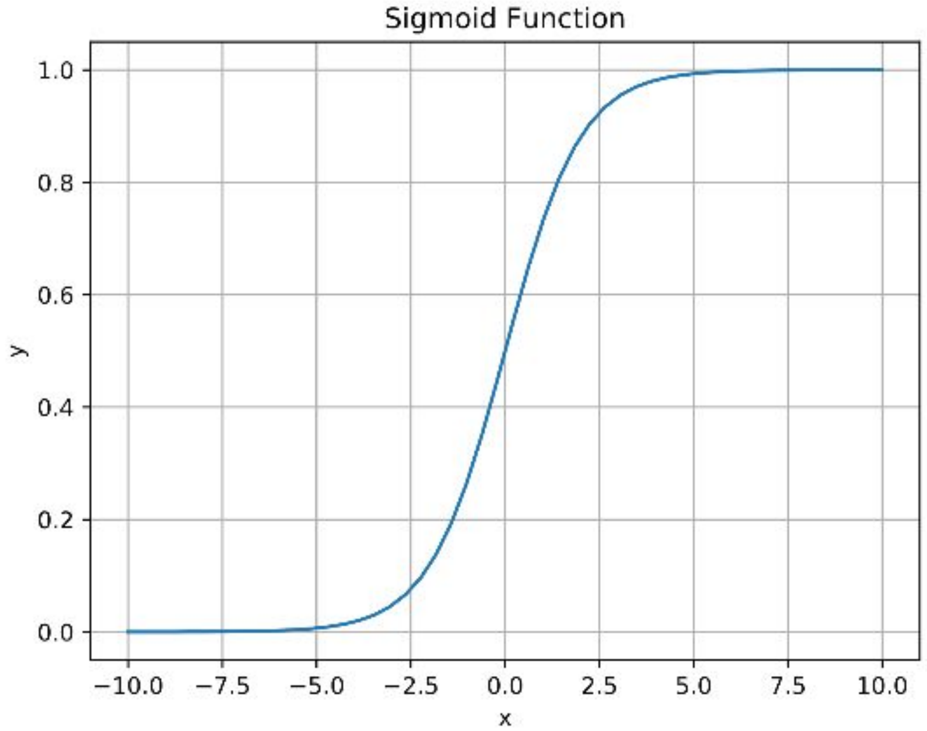
**This sort of initialization helps to set the weight matrix neither too bigger than 1, nor too smaller than 1. Thus it doesn’t explode or vanish gradients respectively.

We define the sigmoid function and its derivative as follows:

```
def sigmoid(x):                          #defining sigmoid function
   x = np.array(x,dtype=np.float128)
def sigmoidprime(x):                     # derivative of sigmoid function
   return sigmoid(x)*(1-sigmoid(x))
```

## Sigmoid Function

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



**For each command, we read the respective sound files, convert them into mfcc format, add labels and store them in a new array

```
data, samplerate = sf.read(b)
   data1 = mfcc(data,samplerate)
   data = data1.reshape(4043,)
```

**Here, you can see that the  original data file is reshaped into a row vector of size 4043.

So,  Input(X) size= 4043*1

**Similarly, with reference to the code,

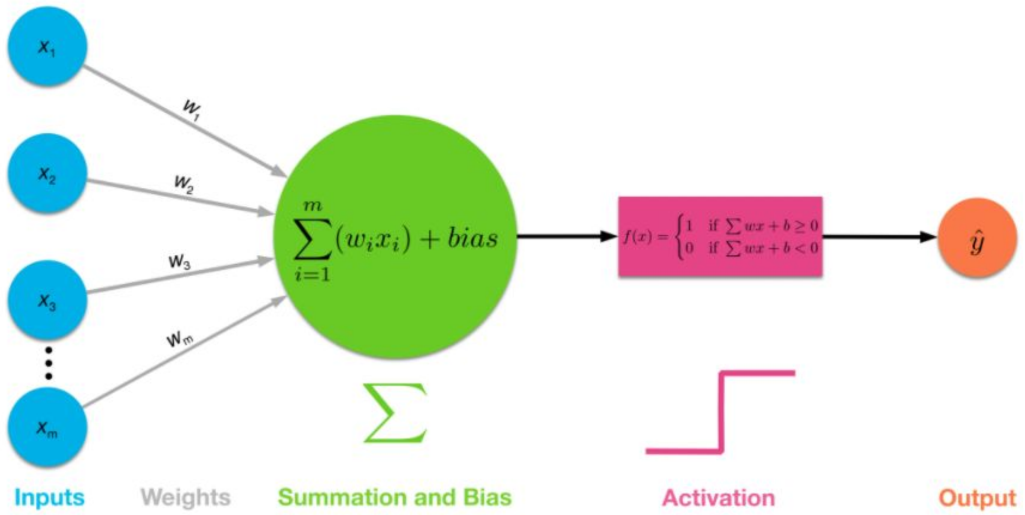
Weighted Matrix(W) size= 4043*5 (most probably to allow matrix multiplication)

Bias matrix(b) size= 5*1

Output matrix(y')= 5*1 (because bot bias matrix and the output matrix must be additive in nature

# Recurrent Neural Network

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. For the purpose of supervised learning(Regression), we are using RNNs here.



$x_1$
$x_2$
$x_3$
$x_m$
$w_1$
$w_2$
$w_3$
$w_m$
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

Single-layer Perceptron Network

## Training the model: Linear regression

output matrix(y')= sigmoid(W.X +b)

y'=f(W,b)

The sigmoid function is defined earlier(tanh in our case)

The cost function is defined using mean square difference:

f(W,b)= (0.5/m)∑(y-y')^2

summed over all the input and output training sets (m= total no. of training sets)

To approximately fit our data in the linear regression model

## Training the model: Logistic Regression Categorial Cross entropy

But for our sigmoid function, logistic regression is to be done as it is a non-linear function so gradient descent algorithm can’t be applied and we have no guarantee of finding a global minimum and hence the cost function is found out as follows:

$$E_t(y_t, \hat{y}_t) = -y_t log(\hat{y}_t)$$
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

ŷ is the predicted value

Categorical crossentropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes

# Training the model: Stochastic gradient descent

Training the algorithm using gradient descent gives the nearest  values of weighted and biased matrices so that they nearly satisfy the given dataset.

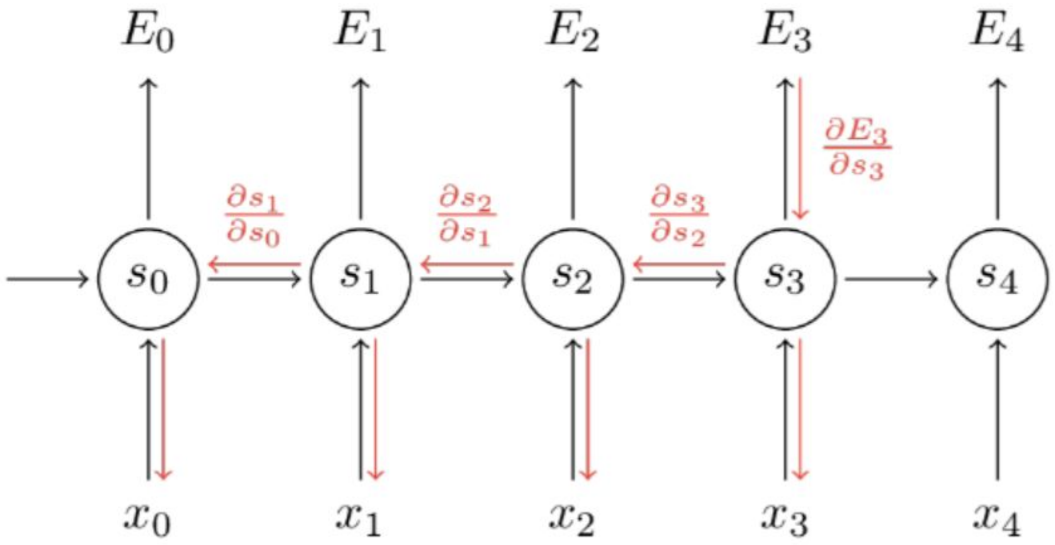
Remember, E is the summation of our cost functions for respective input datasets.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Therefore, E = f(W,b)

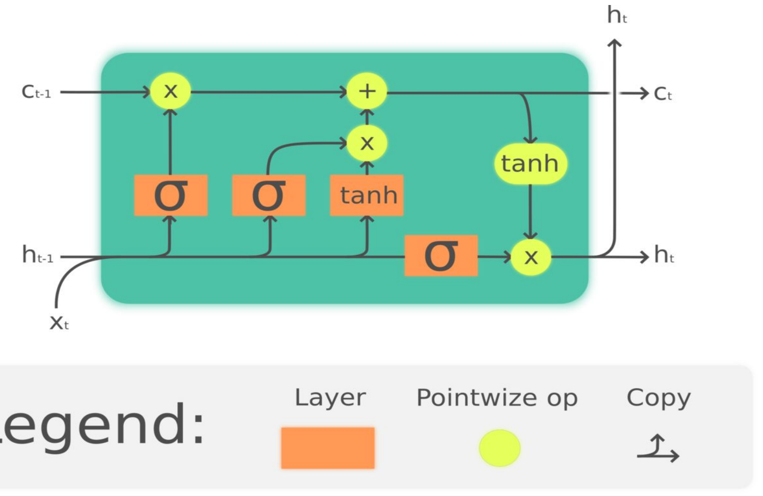
We have to find that pair of W and B for which the gradient descent is minimum

## Back Propogation(Gradient Descent)


$E_0$
$E_1$
$E_2$
$E_3$
$E_4$
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
$s_0$
$s_1$
$s_2$
$s_3$
$s_4$
$x_0$
$x_1$
$x_2$
$x_3$
$x_4$

## Training the model: Long short-term memory

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**.LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs



## Speech Recognition: Classification(Labeling)

An example for label classification of the stop label(As we have already seen that the output matrix is of the order 5*1):

```
for i in range (0,6250):                                    # manually
assigning labels
   y[i][0]=0.0
   y[i][1]=0.0
   y[i][2]=0.0
   y[i][3]=0.0
   y[i][4]=1.0
y4l = np.append(y4,y,axis=1)
print("y4l shape {}".format(y4l.shape))
```

## Speech Recognition: Testing accuracy

After we get the approximate W and b values, we test the training sets and check the accuracy on our datasets.

We have 20,000 conditioned samples out of which 4000 (20%) will be used as testing sets.