EMBEDDEDML

EMBEDDED MACHINE LEARNING IN C

CREATED BY: CHARLES ZALOOM

WHAT IS EMBEDDEDML?

EmbeddedML was first created to be an alternative to the limited options available for Artificial Neural Networks in C. It is designed to be efficient without sacrificing ease of use. It is meant to support students as well as industry experts and it is built to be expandable and straightforward to manipulate.

The core of EmbeddedML is built around a centralized ANN structure that is manipulated using various functions. The core functions of the library consist of model training, and model execution as well as three activation functions; *RELU*, *RELU2*, and *weak_softmax* that are specifically optimized for embedded applications. The other functions can be thought of as layers that can be added or removed depending on the use case. This is what allows the library to perform for both student projects and for projects performing on limited resource systems.

Another unique approach that EmbeddedML takes is that it uses an external helper program to perform the format model initialization. This is done through the Model Designer program. This program takes as an input the desired topology of a network and queries for other details. The output is source code that may be copied into the ML system and initializes a new model or loads a model from a file. By removing the requirement of pre-initializing models by the user, time and effort can be focused on the application and less on EmbeddedML's design.

APPLICATIONS

Intelligent Sensor Systems

Allows for dynamically learning devices that self adapt to the application, environment, and user without the constraints of statically configured systems.

Internet of Things

Improvements in system robustness and security by training with local sensor data sources independently of unreliable, unavailable, or untrustworthy network communication.

Education

Use of EmbeddedML's utilities allow for a gradual introduction into embedded applications of machine learning.

MODEL DESIGNER

Designer.c

- Used to correctly format the model before usage.

Compile: *gcc -o designer designer.c embeddedML.o*

Usage: *./designer 3 6 3 2*

- The program takes the desired ANN topology as input and outputs corresponding C code to implement that model.
- Example:
    - As shown above, the desired topology is 3 6 3 2. This includes 3 input neurons, 6 neurons in the second layer, 3 neurons in the third layer, and 2 output neurons.

Options:

Create/Load ANN

- **New** – Creates brand new model structure.
- **New Embedded** – Creates brand new model structure with pre-initialized weights.
- **Load** – Creates a model structure that is initialized by the *load_ann* function.

Weight Generator

- Generates weights for the inputted topology.
- Generally used for converting a model to an Embedded one.

EMBEDDEDML LIBRARY

COMPILING

1. Navigate to the folder that contains the library.

gcc -c embeddedML.c

2. Let main.c be a file that uses the EmbeddedML Library.

gcc -o main main.c embeddedML.o

3. Execute the program.

./main

ANN STRUCTURE

```
//-----ANN-----
typedef struct {
    float *weights;
    float *dedw;
    float *bias;
    unsigned int *topology;
    unsigned int n_layers;
    unsigned int n_weights;
    unsigned int n_bias;
    float *output;

    float (*output_activation_function)(float);
    float (*output_activation_derivative)(float);
    float (*hidden_activation_function)(float);
    float (*hidden_activation_derivative)(float);

    float eta;      //Learning Rate
    float alpha;    //Momentum Coefficient
} ANN;
```

CORE FUNCTIONS

train_ann(ANN &model , input , expected_output);

- Trains pre-initialized ANN on an input and expected output.
- Example:

```
//TRAINING CYCLES
unsigned int i;
for(i = 0; i < 1000; i++){
    generate_xorand(x, y);
    train_ann(&net, x, y);
}
```

run_ann(ANN &model , input);

- Executes the model as a math function.
- Provides outputs to *model.output*
- Example:

```
//RUNNING A TRAINED NETWORK
run_ann(&net,x0);
int i;
printf("OUTPUT: ");
for(i = 0; i < net.topology[net.n_layers-1]; i++){
  printf("%f ", net.output[i]);
}
printf("\n");
```

init_ann(ANN &model);

- Initializes weights and biases for the model before training.
- This function is part of the generated initialization by the Designer.c

save_ann(ANN &model, “Filename”);

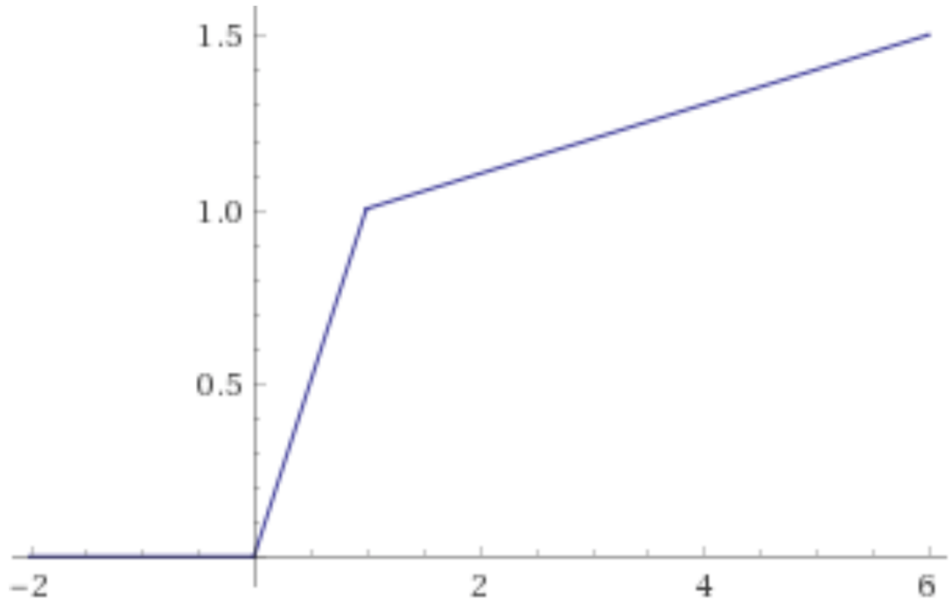
- Saves a model to a file.

load_ann(ANN &model , “Filename”);

- Loads a model from a file.
- Model pre-initialization required. (See *Designer.c)*

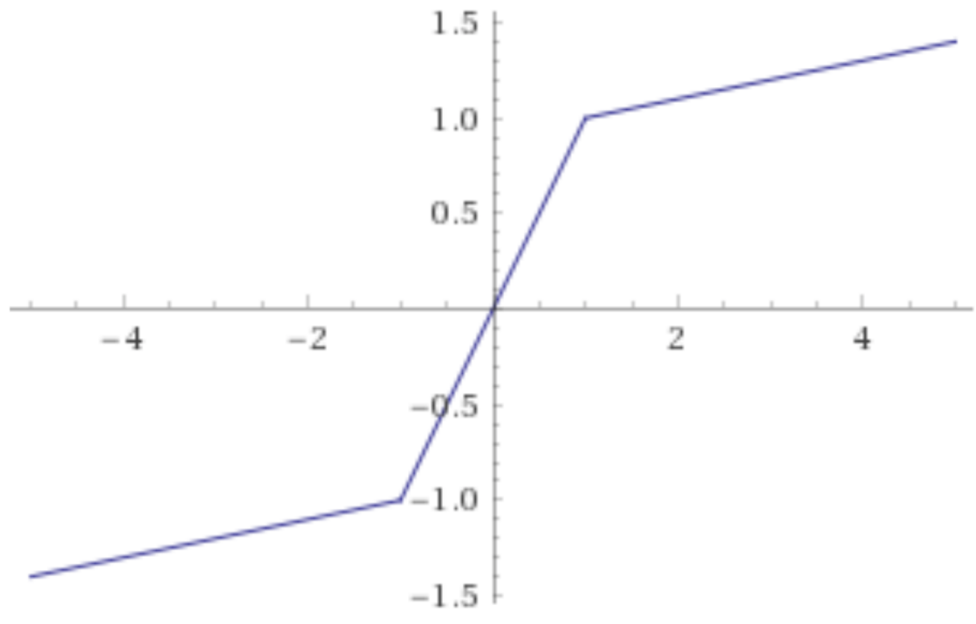
MATHEMATICAL ACTIVATION FUNCTIONS

RELU



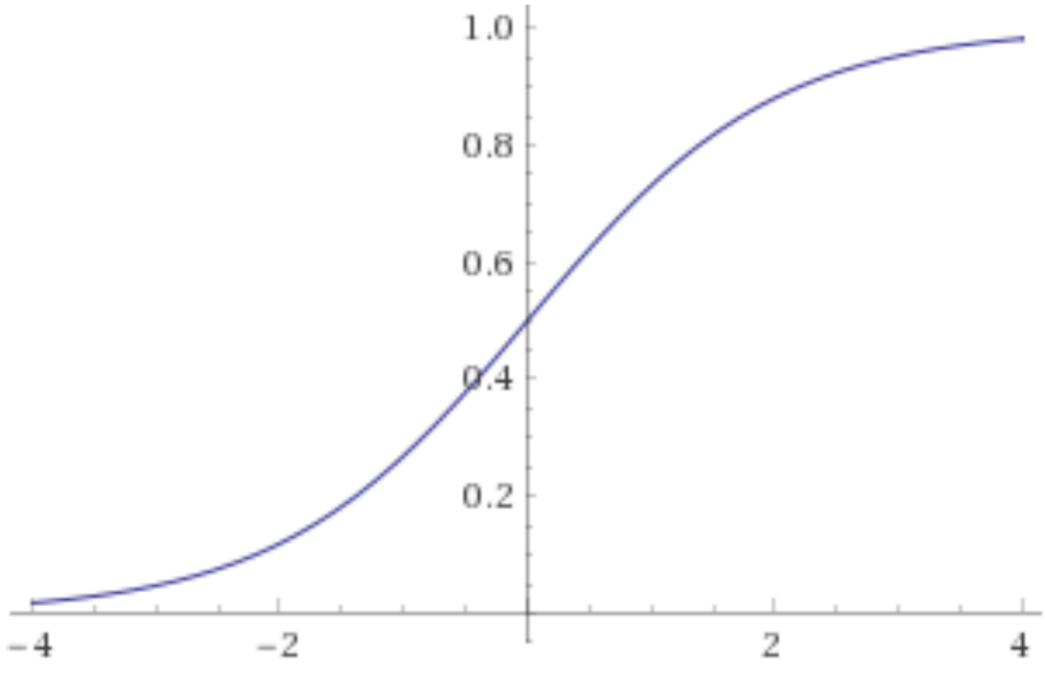
RELU2

- Used for applications with a domain of $-\infty < x < \infty$.



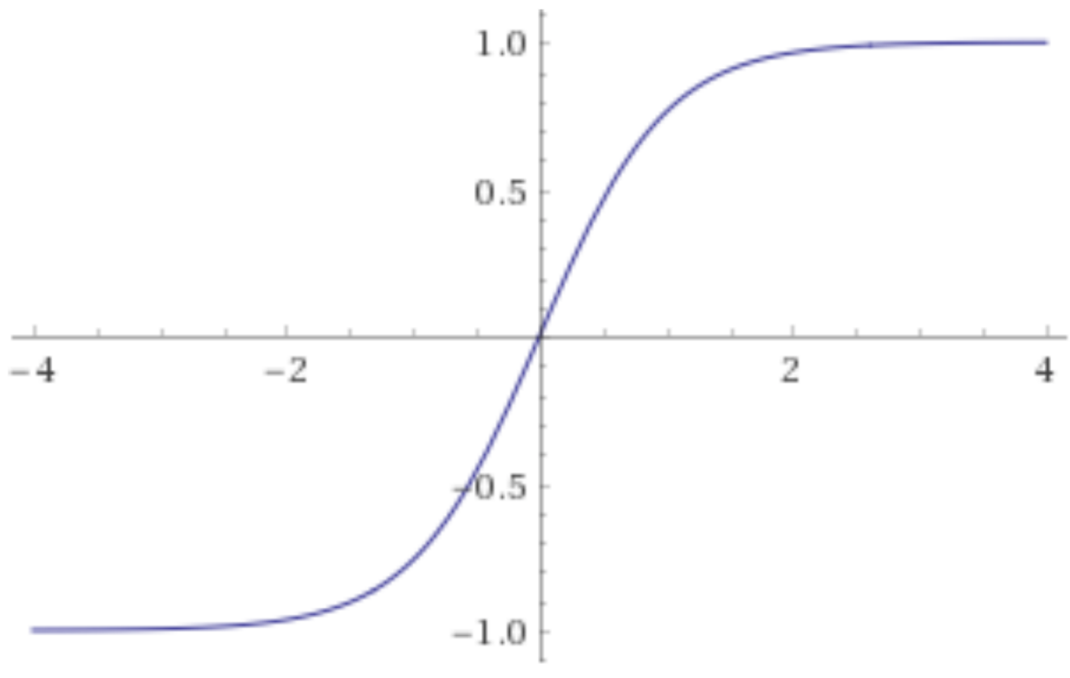
Sigmoid

- Uses *math.h*



Tanh

- Uses *math.h*



UTILITY FUNCTIONS

fill_rand(float *v , unsigned int size, float lower , float upper);

- Fills a vector *v* of length *size* with bound arguments of *lower* and *upper*.

fill_zeros(float *v , unsigned int size);

- Fills a vector *v* of length *size* with zeros.

fill_number(float *v , unsigned int size , float number);

- Fills a vector *v* of length *size* with the argument *number*.

softmax(unsigned int size , float multiplier , float *x , float *y);

- Operates on arguments of an input vector *x* and output vector *y* both of length *size*.
- *multiplier* should be a multiple of 10. This it is used to maintain input values greater than 1.0.

softmax2(unsigned int size , float multiplier , float *x , float *y);

- Same as *softmax*, except the input values are allowed to have negative value.

weak_softmax(unsigned int size , float *x , float *y);

- Note: Requires adjustment based on application.
- Softmax designed for low-precision floating point numbers.
- This uses power of 2 scaling in place of exponential function.

EXAMPLES

Tutorial 1 - Introduction and Basic Usage

Tutorial 2 - Loading Models from File

Tutorial 3 - Usage on Embedded Systems

Tutorial 4 - User Customization

xor_example.c - Example that learns the XOR gate

xor_and_example.c – Example that learns an XOR-AND gate.

CURRENT DEVELOPMENT AND PLANS FOR THE FUTURE

- New Neural Network Models (i.e. RNN, LTSM, Autoencoder, etc…)
- New Neural Network Training methods.
- New Neural Network methods that speed model computation by constructing models from partially trained templates.