Lab Cortex-M4

*Machine Learning (Deep Learning, Artificial Neural Network) using Microcontroller: Embedded AI*
History of AI

A.I. TIMELINE

1950
TURING TEST
Computer scientist Alan Turing proposes a test for machine intelligence. If a machine can trick humans into thinking it is human, then it has intelligence.

1955
A.I. BORN
Term ‘artificial intelligence’ is coined by computer scientist John McCarthy to describe “the science and engineering of making intelligent machines.”

1961
UNIMATE
First industrial robot, Unimate, goes to work at GM replacing humans on the assembly line.

1964
ELIZA
Pioneering chatbot developed by Joseph Weizenbaum at MIT holds conversations with humans.

1966
SHAKEY
The ‘first electronic person’ from Stanford, Shakey, is a general-purpose mobile robot that reasons about its own actions.

A.I. WINTER
Many false starts and dead-ends leave A.I. out in the cold.

1997
DEEP BLUE
Deep Blue, a chess-playing computer from IBM defeats world chess champion Garry Kasparov.

1998
KISMET
Cynthia Breazeal at MIT introduces Kismet, an emotionally intelligent robot insofar as it detects and responds to people’s feelings.

1999
AIBO
Sony launches first consumer robot pet dog AIBO (A.I. robot) with skills and personality that develop over time.

2002
ROOMBA
First mass produced autonomous robotic vacuum cleaner from iRobot learns to navigate and clean homes.

2011
SIRI
Apple integrates Siri, an intelligent virtual assistant with a voice interface, into the iPhone 4S.

2011
WATSON
IBM’s question answering computer Watson wins first place on popular $1M prize television quiz show Jeopardy.

2014
EUGENE
Eugene Goostman, a chatbot passes the Turing Test with a third of judges believing Eugene is human.

2014
ALEXA
Amazon launches Alexa, an intelligent virtual assistant with a voice interface that completes shopping tasks.

2016
TAY
Microsoft’s chatbot Tay goes rogue on social media making inflammatory and offensive racist comments.

2017
ALPHAGO
Google’s A.I. AlphaGo beats world champion Ke Jie in the complex board game of Go, notable for its vast number (2172) of possible positions.
Neural Net CPU

"My CPU is a neural-net processor; a learning computer."
- T-800 - Terminator 2: Judgment Day

The Neural Net CPU is a "learning computer" and one of the most powerful microprocessors ever built. All of the battle units deployed by Skynet contain a Neural Net CPU.

Housed within inertial shock dampers within each battle unit, the CPU gives Skynet the ability to control its units directly, or allow them to function by themselves, learning from a pre-programmed knowledge base as they go. This means that each battle unit has the potential to adapt to its situation, and literally reason through problems and tactical maneuvers. In the case of the various Terminator series, this means that they can learn to behave more like humans in order to be better equipped for infiltration.

It is developed by Miles Bennett Dyson, director of Special Projects at Cyberdyne Systems Corporation, via reverse engineering on the wreckage of a T-800 Terminator in 1984.
ARTIFICIAL INTELLIGENCE IS NOT NEW

Any technique which enables computers to mimic human behavior

MACHINE LEARNING
AI techniques that give computers the ability to learn without being explicitly programmed to do so

DEEP LEARNING
A subset of ML which make the computation of multi-layer neural networks feasible
Neuron

Neuron structure taken from Wikipedia
Biologically Inspired

- Electro-chemical signals
- Threshold output firing
The Perceptron

- Binary classifier functions
- Threshold activation function
The Perceptron: Threshold Activation Function

- Binary classifier functions
- Threshold activation function
Linear Activation functions

- Output is scaled sum of inputs

\[ y = u = \sum_{n=1}^{N} w_n x_n \]
Nonlinear Activation Functions

- Sigmoid Neuron unit function

\[ y_{hid}(u) = \frac{1}{1 + e^{-u}} \]
Nonlinear Activation Functions

- ReLU (Rectified Linear Unit)
Model of a single neuron
Neuron Model

\[ u_k = \sum_{j=1}^{m} w_{kj} x_j \]  
Adder, weighted sum, linear combiner

\[ v_k = u_k + b_k \]  
Activation potential; \( b_k \): bias

\[ y_k = \varphi(v_k) \]  
Output; \( \varphi \): activation function
Layered Networks

Inputs

\( x_1 \)
\( x_2 \)
\( x_n \)

Hidden Neurons

\( y_{h1}(u) \)
\( y_{h2}(u) \)
\( y_{hn}(u) \)

Output Neurons

\( y_{o1}(u) \)
\( y_{o2}(u) \)
\( y_{om}(u) \)

Outputs

\( y_1 \)
\( y_2 \)
\( y_m \)
SISO Single Hidden Layer Network

- Can represent and single input single output functions: \( y = f(x) \)
Training Data Set

- Adjust weights \( \mathbf{w} \) to learn a given target function: \( y = f(x) \)
- Given a set of training data \( X \rightarrow Y \)
Training Weights: Error Back-Propagation (BP)

- Weight update formula:

\[ w(k + 1) = w(k) + \Delta w \]

\[ \Delta w(i) = \eta \frac{\partial e(i)}{\partial w} \]
**Error Back-Propagation (BP)**

**Training error term:** \( e \)

\[
e = \frac{1}{2} (y_{out} - y_{train})^2
\]
Example: The XOR Problem

- Single hidden layer: 3 Sigmoid neurons
- 2 inputs, 1 output

Desired I/O table (XOR):

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Example 2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Example 3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Example 4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: The XOR Problem

- Training error over epoch
### Example: The XOR Problem

- **Mapping produced by the trained neural net:**

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td>0</td>
<td>0</td>
<td>0.0824</td>
</tr>
<tr>
<td>Example 2</td>
<td>0</td>
<td>1</td>
<td>0.9095</td>
</tr>
<tr>
<td>Example 3</td>
<td>1</td>
<td>0</td>
<td>0.9470</td>
</tr>
<tr>
<td>Example 4</td>
<td>1</td>
<td>1</td>
<td>0.0464</td>
</tr>
</tbody>
</table>
Embedded AI Example

- MNIST Data Set
MNIST Data
MNIST Data
Embedded AI Example using MNIST Data Set

0–9 handwritten digit recognition:

MNIST Data maintained by Yann LeCun: http://yann.lecun.com/exdb/mnist/
Keras provides data sets loading function at http://keras.io/datasets
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)

28 x 28 = 784

Number of training examples

Training on PC
Save neural network model
Convert model to C program
Compile and download to target
Neural Network Model

- $n=28 \times 28$
- $m=10$
Deep-Learning Software and Hardware Stack

- Keras
- TensorFlow / Theano / CNTK / ...
- CUDA / cuDNN
- BLAS, Eigen
- GPU
- CPU
Anaconda : Python Data Science Platform

- Copy mnist_mlp.py, send_test.py to C:\work\Anaconda

- Run Spyder
Open mnist_mlp.py

```python
# Trains a simple deep NN on the MNIST dataset.
# Gets to 98.40% test accuracy after 20 epochs
# (there is *a lot* of margin for parameter tuning).
# 42 seconds per epoch on a K520 GPU.
...
7 from __future__ import print_function
8
9 import keras
10 from keras.datasets import mnist
11 from keras.models import Sequential
12 from keras.layers import Dense, Dropout
13 from keras.optimizers import RMSprop

15 batch_size = 128
16 num_classes = 10
17 epochs = 20

19 # the data, split between train and test sets
20 (x_train, y_train), (x_test, y_test) = mnist.load_data()
21 x_train = x_train.reshape(60000, 784)
22 x_test = x_test.reshape(10000, 784)
23 x_train = x_train.astype('float32')
24 x_test = x_test.astype('float32')
25 x_train /= 255
26 x_test /= 255
27 print(x_train.shape[0], 'train samples')
28 print(x_test.shape[0], 'test samples')
```
- **Saved model:** mnist_mlp_model.h5
New STM32 Project: ai
**Minimum Heap Size: 0x2000**
- Enable USART2
- Click Additional Software
- Select Application (4.1.0): SystemPerformance
- Select X-CUBE/Core (4.1.0)
- Then, Click OK
- Click Additional Software and click
Platform Settings and network
- Analyze or Validate on desktop
Creating report file C:\Users\lukas\sta32\cubex\sta32\output\network\validate\report.txt

Complexity/12r error per-layer - nacc=899.948 ron=890.218

<table>
<thead>
<tr>
<th>id</th>
<th>layer [type]</th>
<th>nacc</th>
<th>ron</th>
<th>12r error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>dense_13 (Dense)</td>
<td>50.0%</td>
<td></td>
<td>50.0%</td>
</tr>
<tr>
<td>0</td>
<td>dense_13 nl (Nonlinearity)</td>
<td>0.1%</td>
<td></td>
<td>1.728176e-02 *</td>
</tr>
<tr>
<td>2</td>
<td>dense_14 (Dense)</td>
<td>30.1%</td>
<td></td>
<td>30.1%</td>
</tr>
<tr>
<td>2</td>
<td>dense_14 nl (Nonlinearity)</td>
<td>0.1%</td>
<td></td>
<td>9.645343e-03</td>
</tr>
<tr>
<td>4</td>
<td>dense_15 (Dense)</td>
<td>0.3%</td>
<td></td>
<td>3.0%</td>
</tr>
<tr>
<td>4</td>
<td>dense_15 nl (Nonlinearity)</td>
<td>0.2%</td>
<td></td>
<td>5.334938e-03</td>
</tr>
</tbody>
</table>

Using TensorFlow backend.
Generate Code and Build

CDT Build Console [ai]

777552 1808 11290 790560 c1020 ai.elf
Finished building: default.size.stdout

Finished building: ai.list

10:29:10 Build Finished. 0 errors, 0 warnings. (took 29s.742ms)
Trained weight

```c
#include "network_data.h"

ai_handle ai_network_data_weights_get(void)
{
    static const ai_u8 s_network_weights[690216] = {
        0x4e, 0xb1, 0x15, 0xbf, 0x8d, 0xa5, 0x14, 0xbf, 0x61, 0x9a,
        0x13, 0xbf, 0xcb, 0x8c, 0x12, 0xbf, 0x75, 0x83, 0x11, 0xbf,
        0xff, 0x77, 0x10, 0xbf, 0x88, 0x6c, 0x0f, 0xbf, 0x12, 0x61,
        0x0e, 0xbf, 0x9c, 0x55, 0x0d, 0xbf, 0x26, 0x4a, 0xc0, 0xbf,
        0x4a, 0xb0, 0xb0, 0xbf, 0x39, 0x33, 0xa0, 0xbf, 0xc3, 0x27,
        0x90, 0xbf, 0x8f, 0x7b, 0x08, 0xbf, 0xf6, 0xfa, 0x06, 0xbf,
        0x50, 0x05, 0x66, 0x7b, 0x0f, 0xbf, 0x05, 0xbf, 0x73, 0x0e,
        0x93, 0xbf, 0x47, 0xc4, 0x02, 0xbf, 0x87, 0x7d, 0x01, 0xbf,
        0x11, 0xc0, 0x00, 0xbf, 0x35, 0x81, 0xff, 0xbe, 0x48, 0x6a,
        0xfd, 0xbe, 0x5b, 0x53, 0xfb, 0xbe, 0x6f, 0xc3, 0xf9, 0xbe,
        0xe6, 0x6a, 0xf7, 0xbe, 0xad, 0x0a, 0xf5, 0xbe, 0x5c, 0xf5,
        0xf2, 0xbe, 0xbd, 0xe0, 0xf0, 0xbe, 0x5b, 0x54, 0xef, 0xbe,
        0x32, 0x78, 0xed, 0xbe, 0x59, 0x66, 0xea, 0xbe, 0x98, 0x19,
        0xe8, 0xbe, 0x44, 0x62, 0xe5, 0xbe, 0x6d, 0x92, 0xe4, 0xbe,
        0x52, 0x87, 0xc2, 0xbc, 0x9f, 0x47, 0xe0, 0xbe, 0xad, 0xc0,
        0xdd, 0xbe, 0x67, 0xf0, 0xdc, 0xbe, 0xea, 0xa5, 0xda, 0xbe,
        0x9b, 0xe0, 0xd7, 0xbe, 0x8c, 0xdf, 0xdb, 0xe0, 0x20, 0x5b,
        0xd4, 0xbe, 0xc7, 0x55, 0xdb, 0xe0, 0x05, 0x31, 0xc0, 0xbe,
        0xad, 0x92, 0xcd, 0xbe, 0x2b, 0x49, 0xcb, 0xbe, 0xe8, 0xd2,
        0xc8, 0xbe, 0x46, 0xf8, 0xc0, 0xbe, 0xb4, 0x16, 0xc5, 0xbe,
        0x64, 0xdb, 0x2c, 0xbe, 0xb0, 0x3a, 0xc0, 0xbe, 0xd2, 0xb5,
        0xbe, 0xbe, 0x3f, 0xad, 0xbc, 0xbe, 0x8e, 0x93, 0xba, 0xbe,
        0x31, 0x2b, 0x8b, 0xe0, 0x07, 0x8f, 0xf6, 0xe8, 0x47, 0x1d
    }
}
```
- Open `aiSystemPerformance.c`
- Find from Edit menu: “Fill input” or go to the line 912
Modify aiSystemPerformance.c

```c
/* Fill input vector */

unsigned char string[28 * 28][3];
ioGetUint8((unsigned char*)string, 28 * 28 * 3, 5000);
for (ai_size i = 0; i < 28 * 28; i++) {
    if (string[i][0] == ' ') string[i][0] = '0';
    if (string[i][1] == ' ') string[i][1] = '0';
}
const ai_buffer_format fmt = AI_BUFFER_FORMAT(&ai_input[0]);
for (ai_size i = 0; i < AI_BUFFER_SIZE(&ai_input[0]); ++i) {
    /* uniform distribution between -1.0 and 1.0 */
    /*//const float v = 2.0f * (ai_float) rand() / (ai_float) RAND_MAX - 1.0f;
    const float v = (100.0f*(ai_float)(string[i][0] - 0x30) + 10.0f*(ai_float)
        (string[i][1] - 0x30) + (ai_float)(string[i][2] - 0x30)) / 255.0f;
 */
    batch = ai_mnetwork_run(net_exec_ctx[idx].handle, &ai_input[0], &ai_output[0]);
    if (batch != 1) {
        ailogErr(ai_mnetwork_get_error(net_exec_ctx[idx].handle),
            "ai_mnetwork_run");
        break;
    }
}
unsigned char recognized_digit;
ai_float out_data_float[10];
for (int i = 0; i < 10; i++) out_data_float[i] = *(ai_float *)(out_data + i * 4);
recognized_digit = 0;
for (int i = 0; i < 10; i++) if (out_data_float[i] > out_data_float[recognized_digit]) recognized_digit = i;
printf("%d", recognized_digit);
tend = cyclesCounterEnd();
```
- Build and Download to the target board
- Connect serial cable and check COM port number
Run send_test.py

- Press RESET button (black button) and run send_test.py

Recognized digit: 7.

Recognized digit: 5.

Recognized digit: 4.
Recognized digit: 2.

Results for "network", 16 inferences @168MHz/168MHz (compile duration: 52.216 ms (average),
CPU cycles: 8772339 -75/+87 (average,-/+),
CPU Workload: 5%
cycles/MACC: 13 (average for all layers)
used stack: NOT CALCULATED
used heap: 0:0 0:0 (req:allocated,req:released) cfg=0
Exercise

- 주어진 예제를 실행하여 메모리 사용량 및 속도를 검토하고 실제 인식 성능을 확인한다.

- 모델에서 코드 생성 시 컴프레션을 하는 이유는 메모리 사용량을 줄여서 프로그램 메모리에 탑재 가능하도록 하기 위함이다. 뉴럴네트워크 모델에서 뉴런의 숫자를 줄이면 대신 컴프레션을 하지 않아도 메모리에 탑재가 가능하게 될 수 있다. 뉴런의 개수를 줄여서 컴프레션을 하지 않고도 프로그램 메모리에 탑재할 수 있도록 한다. 이때, 뉴런을 줄이지 않고 컴프레션을 한 경우와, 뉴런을 줄여서 컴프레션을 한 경우의 체감 인식 성능을 비교해 본다.