Lab Cortex-M4

Machine Learning (Deep Learning, Artificial Neural Network) using Microcontroller : Embedded AI
Human Brain vs Computer
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 AI 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 (A1 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 AI AlphaGo beats world champion Ke Jie in the complex board game of Go, notable for its vast number \(2^{192}\) 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 it's 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

ARTIFICIAL INTELLIGENCE
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 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

![Step Threshold Diagram]
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

\[
\begin{align*}
\text{Inputs} & \quad \text{Hidden Neurons} & \quad \text{Output Neurons} & \quad \text{Outputs} \\
X_1 & \rightarrow & y_{h1}(u) & \rightarrow y_{o1}(u) & \rightarrow Y_1 \\
X_2 & \rightarrow & y_{h2}(u) & \rightarrow y_{o2}(u) & \rightarrow Y_2 \\
\vdots & \rightarrow & \vdots & \rightarrow \vdots & \rightarrow \vdots \\
X_n & \rightarrow & y_{hn}(u) & \rightarrow y_{om}(u) & \rightarrow Y_m
\end{align*}
\]
SISO Single Hidden Layer Network

- Can represent and single input single output functions: $y = f(x)$
Adjust weights \((w)\) 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:

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

\[ \Delta w(i) = \eta \cdot \frac{\partial e(i)}{\partial w} \]
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
Embedded AI Example using MNIST Data Set

0–9 handwritten digit recognition:

MNIST Data maintained by Yann LeCun: [http://yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/)
Keras provides data sets loading function at [http://keras.io/datasets](http://keras.io/datasets)
Training

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

- Training on PC
- Save neural network model
- Convert model to C program
- Compile and download to target

Number of training examples: 28 x 28 = 784
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

- 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).
# 2 seconds per epoch on a K520 GPU.

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical

batch_size = 128
num_classes = 10
epochs = 2

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784)
x_train = x_train.astype('float32') / 255
x_test = x_test.reshape(10000, 784)
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Python 3.7.3 (default, Mar 27 2019, 17:13:21) [GCC 7.3.0] -- An enhanced Interactive Python.

In [1]:

In [2]:
- Saved model: mnist_mlp_model.h5
Open STM32Cube and Select New Project
Select Board and Start Project
Pinout & Configuration

- Enable USART2
Additional Software

- Select Application(3.3.0): SystemPerformance
- Select X-CUBE/Core(3.3.0)
- Then, Click OK
- Select X-CUBE-AI and check AI Core, AI Application
- Then Click Add model
- Keras, Saved model, mnist_mlp_model.h5
- Compression: 4
- Click Output from Window menu
- Click Analyze
- Click Validate on desktop

Matching criteria: L2 error < 0.01 on the output tensor

- Ref layer 0 matched with C layer 1, error: 0.01565543
- Ref layer 2 matched with C layer 3, error: 0.0061046
- Ref layer 4 matched with C layer 5, error: 0.0066665824

Validation: OK
Validation OK
Python validation ended
- Platform Settings
- COM Port: USART2
Project Manager

- Project Name: ai, Project Location: C:\emwork
- Toolchain/IDE: Other Toolchains (GPDDC)
- Minimum Heap Size: 0x2000
- Save Project
- GENERATE CODE
Open Visual Studio and start VisualGDB project

- Location: C:\emwork
- Name: ai
- Click OK
- Project file: C:\emwork\ai.gpdsc
- Floating point support: Hardware
JTAG programmer: ST-Link v2.1
Project Property
Add Include directories

- Add list by clicking + sign
Add the directories

- C:\emwork\ai\Middlewares\ST\AI\AI\include
- C:\emwork\ai\Middlewares\ST\AI\AI\data
- C:\emwork\ai\Middlewares\ST\Application\SystemPerformance\Inc
Add source files and libraries files
Source files and Library files

- C:\emwork\ai\Middlewares\ST\AI\AI\src\network.c
- C:\emwork\ai\Middlewares\ST\AI\AI\data\network_data.c
- C:\emwork\ai\Middlewares\ST\Application\SystemPerformance\Src\aiSystemPerformance.c

- C:\emwork\ai\Middlewares\ST\AI\AI\lib\network_runtime.a
- C:\emwork\ai\Drivers\CMSIS\Lib\GCC\libarm_cortexM4lf_math.a
Trained weight
- Open aiSystemPerformance.c
- Quick find: “Fill input”
Modify aiSystemPerformance.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';
}
for (ai_size i = 0; i < aiBufferSize(&ai_input[0]); ++i) {
    /* uniform distribution between -1.0 and 1.0 */
    /*in_data[i] = 2.0f * (ai_float) rand() / (ai_float) RAND_MAX - 1.0f;
    in_data[i] = (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_ctx[idx].handle, &ai_input[0], &ai_output[0]);
    if (batch != 1) {
        aiLogErr(ai_mnetwork_get_error(net_ctx[1].handle),
            "ai_mnetwork_run");
        break;
    }
    unsigned char recognized_digit;
    recognized_digit = 0;
    for (int i = 0; i < 10; i++) if (out_data[i] > out_data[recognized_digit]) recognized_digit = i;
    printf("%d", recognized_digit);
    tend = dwtGetCycles() - tstart;
- Build

The screenshot shows a build process in Microsoft Visual Studio. The build output indicates that the build succeeded with no errors or warnings. The build report mentions the utilization of memory resources:

- Flash: 737KB out of 1024KB (72%)
- SRAM: 10KB out of 128KB (8%)
- CodeRAM: 0 bytes out of 64KB (0%)

The build summary confirms that all builds succeeded.
Download to the target board and run
- Connect serial cable and check COM port number
Run send_test.py

Recognized digit: 7.

Recognized digit: 5.

Recognized digit: 4.
Recognized digit: 2.

Results for "network", 16 inferences @168MHz/168MHz (compilation)
  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
주어진 예제를 실행하여 메모리 사용량 및 속도를 검토하고 실제 인식 성능을 확인한다.

메모리 사용량을 줄이기 위하여 모델을 축소하여 실행해 본다. (레이어 수 또는 유닛 수 축소) 메모리 사용량, 실행 속도, 인식 성능을 비교해 본다. 파이선 프로그램의 인식 성능 수치의 변화가 실제 인식 성능에서 체감할 만한 변화가 있는지 알아본다.