Embedded Machine Learning

Using STM32CubeIDE and X-CUBE-AI Extension.

for model integration and code generation.

Deploying a Pretrained Text Classification Keras Model.

optimized for embedded systems and validated on the STM32 NUCLEO-H723ZG board

Achieving Inference Parity with Python

by comparing outputs between the STM32 and Python runtime environments using predefined inputs .

Creating the STM32 Project

- Launch New STMCubeIDE or CubeMX Project
- Software Updates for X-CUBE-AI
- Select Board, Generate New Project
- Accept Defaults
- Increase Stack and Heap
- Select X-CUBE-AI from Middleware

Importing the “Dense” Model

- Go the Middleware, select X-CUBE-AI
- Click + network
  - Browse to text_classification_dense_input.h5
- Run “Analyze”
- Save the .ios or “Save All”
- Yes to save, yes to generate code.

Build, Configure UART for Printing

- Optional Analyzing on Desktop, Target

Make sure you add clock init (see `MX_USART3_UART_Init`)

```
__HAL_RCC_USART3_CLK_ENABLE();
```

- Sync settings with Virtual COM Port in Device Manager
- Same with serial terminal reader (Terminal, putty, etc.)
- Compare both to MX_USART3_UART_Init settings
- Project Properties, C/C++ Build Settings
  - mcu/mpu settings, use float with printf with nano-newlib

X-CUBE-AI Generated Code and Reports

- Project Explorer, X-CUBE-AI, App
- network_generate_report.txt
- Note in the report, output dir, verbose model format
- app_x-cube-ai.c network.c network_config.h network_data.h network_data_params.h
- app_x-cube-ai.h network.h network_data.c network_data_params.c network_generate_report.txt
- Doxygen view of the network framework codegen next

X-CUBE-AI CodeGen: What the Network framework “Does”

- Doxygen view and X-CUBE-AI-Framework.pdf under X-CUBE-AI/App
- Initializes the AI Network: Sets up network parameters, input and output shapes, and loads weights for inference.
- Manages Memory Buffers: Allocates and deallocates memory for model inputs, outputs, and intermediate layers.
- Handles Model Inference: Executes the neural network by feeding inputs through layers to generate predictions.
- Configures Data I/O: Prepares data in the correct format for the model and retrieves outputs for further processing.
- Provides Debug and Profiling Support: Includes hooks for performance monitoring and debugging of inference runs.

Pre-Processing Step with Python

- Run Python Script: Execute the run_inference.py to generate preprocessed float values for each text example.
- Observe Output and Mappings: Display the preprocessed values and compare them with the expected sentiment scores.
- Explain Simplified Vocabulary: Highlight that the vocabulary is manually defined and simpler than the original TensorFlow tutorial.
- No TensorFlow Tokenizer in Embedded Code: Due to resource constraints, the full TensorFlow tokenizer can’t be used directly in STM32.
- Use Python for Input Floats: Python is used to map text inputs to float values, which are then fed directly into the model on STM32.

Editing the “USER CODE”

- Limited to one input/output at a time per AI_NETWORK_IN_NUM, AI_NETWORK_OUT_NUM, both defined as “1” dimension
- Grab the float “indices” and create an array, e.g.
  - `const float predefined_inputs[] = {0.8f, 0.5f, 0.2f};`
  - Edit `int acquire_and_process_data(ai_i8* data[])`
  - Edit `int post_process(ai_i8* data[])`
  - Review X-CUBE-AI/App/app_x-cube-ai.c

Build, Run

- Build, Debug
- Breakpoint in post_process
- Setup Terminal

Recap

- Create New Project in STM32CubeIDE
- Select NUCLEO-H723ZG board and enable X-CUBE-AI middleware.
- Import the Dense Model (text_classification_dense_input.h5)
- Navigate to the X-CUBE-AI tab, create a new network, and select the model file.
- Analyze and Generate Code
- Run the analysis to verify compatibility, save configuration, and generate the necessary code files.
- Configure Project Properties and UART
- Increase stack and heap size.
- Set up UART for serial printing.

Recap (cont.)

- Run Python Script for Preprocessing
- Use run_inference.py to map text inputs to float values.
- Simplified Vocabulary and Indices
- Predefined float values (0.8, 0.5, 0.2) used instead of TensorFlow tokenizer.
- Mapping Floats to Sentiments
- 0.8 → Positive sentiment
- 0.5 → Neutral sentiment
- 0.2 → Negative sentiment

Contribute

- Please provide feedback in the form of Issues
- https://github.com/stevemac321/EmbeddedMachineLearning
- Links in the description of the Video