# MCU 기반 플랫폼에서 인공지능 학습을 위한 RNN 모델 구현

마준익, 곽준호, 조정훈*

경북대학교 전자공학부

macs6848@knu.ac.kr, junho7513@knu.ac.kr, *jcho@knu.ac.kr

# Implementation of an RNN Model for Artificial Intelligence Training on an MCU-Based Platform

Ma Junik, Kwak Junho, Cho Jeonghoon*

School of Electronics Engineering. Kyungpook National Univ.

요 약

본 논문은 MCU(Micro Controller Unit) 기반 플랫폼에서 RNN(Recurrent Neural Network) 모델을 구현하고, 추론과 학습 과정을 진행한 연구 결과를 발표한다. 이 연구는 자원이 제한된 MCU 환경에서도 인공지능 모델의 학습과 추론이 가능함을 보여주며, MCU를 사용하는 기기들에 대한 개인화된 경험을 제공할 수 있는 가능성을 탐구한다. 연구 과정에서는 TensorFlow에서 구현된 모델을 분석하여 핵심 요소만을 선별하고, 이를 C언어로 재구현함으로써 메모리 사용량과 연산 오버헤드를 최소화하였다. 실험 결과는 그림 4와 그림 5를 통해 제시되었으며, 서버급 환경에서의 TensorFlow 실행 결과와 비교했을 때 매우 유사한 손실 감소율을 달성하면서도 메모리 사용량 면에서는 뚜렷한 차이를 보여준다.

## Ⅰ. 서 론

인공지능에 대한 관심이 높아지면서, MCU를 이용한 인공지능 애플리케이션의 활용도 점차 증가하고 있다. 그러나 인공지능 모델은 일반적으로 높은 연산량과 상당한 메모리를 요구하기 때문에, 자원이 제한된 MCU 환경에서는 서버급 환경에서 사용되는 모델을 그대로 적용하는 것이 어렵다. 대부분의 경우, MCU에서는 사전에 학습된(pre-trained) 모델을 사용하여 추론 과정만을 수행할 수 있지만 [1], 학습 과정을 진행하지 못하기 때문에 이러한 방식은 개인화된 경험을 제공하기 어렵다. 따라서, 최소한의 기능만을 사용하여 메모리 사용량을 줄인, MCU 상에서도 학습이 가능한 모델을 개발하고자 하였다. 연구에서는 시계열 데이터를 사용하는 RNN 모델을 구현하였다. 본 논문에서는 MCU에서의 인공지능 학습에 대한 실행 가능성을 보여주며, 인공지능 기술의 더욱 광범위한 적용 가능성을 제시한다.

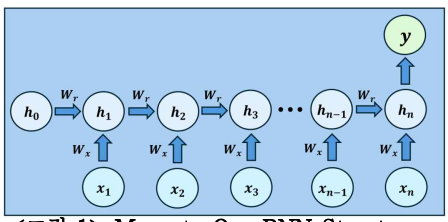
## Ⅱ. 배경지식

인공지능에서는 입력 데이터가 주어질 때, 해당 데이터는 모델의 가중치와 함께 연산되어 결과를 출력하는 과정을 추론(inference)이라고 한다. 이 과정은 순전파(forward propagation) 단계에서 이루어진다. 추론을 통해 생성된 출력값과 실제 값(레이블)의 차이는 손실(loss)로 표현되며, 이 손실을 최소화하는 방향으로 가중치를 조정하는 과정을 학습(learning)이라고 부른다. 학습 과정은 역전파(backpropagation)를 통해 이루어지며, 초기에 무작위로 설정된 가중치들은 점차 최적화되어 간다.

본 연구에서는 시계열 데이터를 효과적으로 학습할 수 있는 RNN(Recurrent Neural Network) 모델을 구현하였다. RNN은 과거의 정보가 현재의 출력에 영향을 미치는 구조로 설계되어 있어, 시간에 따라 변화하는 데이터나 시퀀스 데이터를 처리하는 데 적합하다. 이러한 특성은 시간에 따라 변화하는 자연 현상이나 언어 등의 시계열 데이터를 학습하는 데 특히 유용하다.

## Ⅲ. 본론

이 연구에서는 메모리 사용과 연산 오버헤드를 최소화하기 위해 TensorFlow에서 구현된 인공지능 모델을 분석하였다. 분석을 통해 추론과 학습 과정에 필수적인 요소들만을 선별하여, 이를 C언어로 구현한 RNN 모델을 개발하였다. 본 연구에서는 시계열 데이터 처리에 적합한 RNN 구조를 채택하였으며, 특히 다수의 입력 시퀀스로부터 단일 출력 결과를 도출하는 'Many to One' 방식의 RNN을 사용하였다.




<그림 1> Many to One RNN Structure

그림 1에서는 RNN 구조가 어떻게 구현되었는지 보여준다. 여기서 $x_{n-1}$의 결과는 $h_{n-1}$에 저장되며, 다음 입력 $x_n$은 $h_{n-1}$의 정보를 반영하여 새로운 출력 $h_n$을 출력한다. 이러한 방식으로 RNN의 연속적인 정보처리 능력을 구현하여, 이전 상태의 정보가 현재 상태의 출력에 영향을 미치도록 설계하였다.

$$h_n = \tanh(W_x x_n + W_r h_{n-1} + b_n) \quad (1)$$

$$i_n = W_x x_n + W_r h_{n-1} + b_n \quad (2)$$

$$h_n = \tanh(i_n) \quad (3) \quad [2]$$

수식 (1)은 $h_n$에 대한 정의이며 이를 간소화하기 위해 수식 (2)와 (3)으로 분리하였다. 역전파를 진행하기 위해 경사하강법과 체인룰을 사용하며, 이 과정에서 각 가중치의 영향력을 평가하기 위해 $h_n$과 $i_n$에 대한 gradient를 계산해야 한다.

$h_n$과 $i_n$에 대한 gradient는 각각 수식 (4), (5)이다.

$$gh_n = \frac{\delta L}{\delta h_n} \quad (4)$$

$$gi_n = \frac{\delta L}{\delta h_n}\frac{\delta h_n}{\delta i_n} = gh_n \tanh'(i_n) = gh_n(1-\tanh^2(i_n)) \quad (5)$$

$$g W_x = \frac{\delta L}{\delta i_n}\frac{\delta i_n}{\delta W_x} = gi_n x_n \quad (6)$$

$$g W_r = \frac{\delta L}{\delta i_n}\frac{\delta i_n}{\delta W_r} = gi_n h_{n-1} \quad (7)$$

$$gh_{n-1} = \frac{\delta L}{\delta i_n}\frac{\delta i_n}{\delta h_{n-1}} = gi_n W_r \quad (8)$$

$$gb_n = \frac{\delta L}{\delta i_n}\frac{\delta i_n}{\delta b_n} = gi_n \quad (9)$$

$$g W_x = gi_n x_n + gi_{n-1}x_{n-1} + \cdots + gi_1 x_1 \quad (10)$$

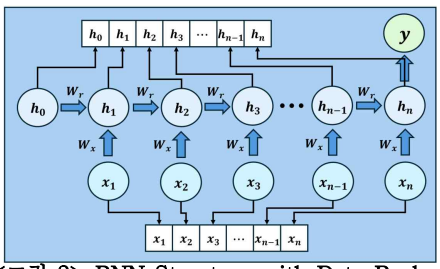
$$g W_r = gi_n h_{n-1} + gi_{n-1}h_{n-2} + \cdots + gi_1 h_0 \quad (11)$$

$$gb = gi_n + gi_{n-1} + \cdots + gi_1 \quad (12)$$

$$gx_n = gi_n W_x \quad (13)$$

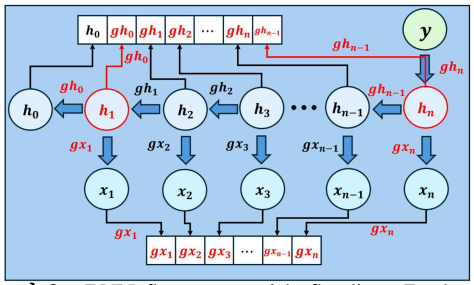
수식 (6), (7), (8), (9)는 각 파라미터들의 gradient에 대한 수식이다. $W_x$와 $W_r$, $b$는 1부터 n까지의 gradients의 합으로 이루어지기 때문에 수식 (10), (11), (12)로 나타낼 수 있다.

역전파를 진행하기 위해서는 순전파 과정에서 $h$와 $x$의 값을 별도로 저장해야 한다는 점을 위의 수식을 통해 확인할 수 있다. 이를 통해 역전파를 진행할 때 각 가중치에 대한 업데이트를 효과적으로 수행할 수 있다.



<그림 2> RNN Structure with Data Backup

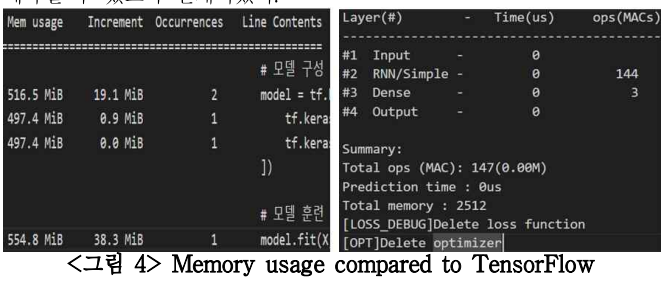
역전파는 순전파 과정이 완료된 후, n부터 1까지의 순서로 각 가중치를 업데이트해 나가는 과정이다. 역전파를 진행하는 동안 순전파에서 저장해 두었던 $h$와 $x$의 값을 활용하며, 이 값들을 사용한 후에는 더 이상 필요하지 않은 $h$와 $x$의 메모리 공간에 gradient 값을 덮어써 메모리 사용을 최소화한다. 이 방법으로 모든 $h$와 $x$에 대한 gradient 값을 계산한 후, 수식 (10), (11), (12)에 대한 연산을 수행한다. 해당 결과를 바탕으로 Optimizer를 통해 $W_x$, $W_r$, $b$를 업데이트한다. ($x$에 대한 gradient 값(수식 (13))은 이전 layer를 업데이트할 때 사용되고, 해당 연구에서 Optimizer는 Adam Optimizer를 사용하였다.)



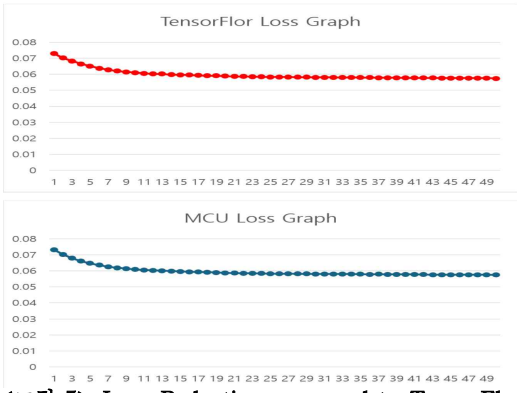
<그림 3> RNN Structure with Gradient Backup

## Ⅳ. 연구결과

이 연구에서는 흑점 데이터셋을 입력 데이터로 사용하였으며, 매월 흑점의 수를 측정한 데이터이다. 이 모델은 12개월 동안의 흑점 수를 기반으로 다음 달의 흑점 수를 예측하는 구조로, 이는 12개의 입력 데이터가 하나의 출력 결과를 생성하는 RNN의 'Many to One' 방식을 따른다. 이 방식은 시계열 데이터의 이전 상태를 고려하여 미래의 값을 효과적으로 예측할 수 있도록 설계되었다.




<그림 4> Memory usage compared to TensorFlow




<그림 5> Loss Reduction compared to TensorFlow

## Ⅴ. 결론

본 논문에서는 태양의 흑점 데이터를 활용하여 미래의 흑점 수를 예측하는 RNN 모델을 MCU 상에서 실행할 수 있도록 구현하고, 이 모델의 추론 및 학습 과정을 수행한 연구 결과를 발표하였다. 그림 4와 그림 5를 통해 서버급 환경에서 Tensorflow를 사용하여 수행한 결과와 매우 유사한 손실 감소율을 달성했음을 확인할 수 있으며, 메모리 사용량 측면에서도 뚜렷한 차이를 보여준다. 하지만 RNN 모델은 기울기 소실 문제로 인한 한계가 있기 때문에 향후 LSTM이나 GRU와 같은 보다 발전된 모델에 대한 연구도 계속 진행될 필요가 있다.


## ACKNOWLEDGMENT

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 1711160343, 차량 ECU 응용소프트웨어 개발 및 검증자동화를 위한 가상 ECU기반 차량레벨 통합 시뮬레이션 기술개발).


## 참 고 문 헌


[1] Q. Liang, P. Shenoy and D. Irwin, "AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures," 2020 IEEE International Symposium on Workload Characterization (IISWC), Beijing, China, 2020, pp. 145-156

[2] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, Yanbo Gao; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 5457-5466