Implementação de uma rede neural em um microcontrolador STM32

Esse relatório busca explicar como implementar uma rede neural em um microcontrolador STM32. Ele pode ser dividido nas seguintes partes:

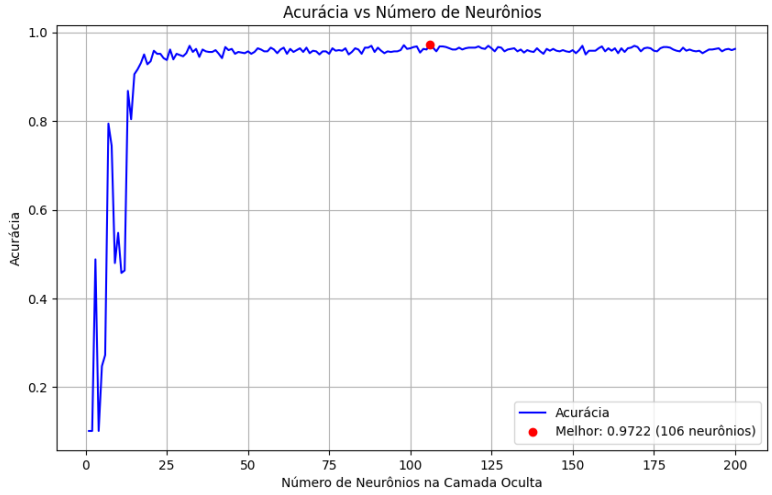
- Treinamento da rede neural;
- Conversão para linguagem C utilizando emlearn;
- Código em python para enviar o dataset via serial;
- Código do microcontrolador em C para receber o dataset, calcular o resultado e retornar via serial;

Treinamento da rede neural

Para esse trabalho foi utilizado o dataset *Optical Recognition of Handwritten Digits*. O dataset consiste em uma matrix 8x8 (64 entradas) com valores de 0-15 onde cada atributo representa a cor de um pixel. As classes resultantes são os números de 0 a 9. Cada grupo possui aproximadamente 180 amostras.

Após carregar, o dataset foi divido em conjunto de teste (20%), validação (40%) e treinamento (40%).

Com os dados prontos, foi feito um teste para descobrir qual é o melhor número de neurônios na camada oculta. Para isso foi iterado entre 1 e 200 neurônios e verificada a taxa de acerto contra o conjunto de validação. O resultado é o seguinte:



Como a partir de aproximadamente 25 neurônios os resultados ficaram bem consistentes, optei por gerar uma tabela com os 10 melhores resultados e a quantidade de neurônios:

Posição	Taxa de acerto	Neurônios
1º	0.9722	106
2º	0.9708	98
3º	0.9694	169
4º	0.9694	153
5º	0.9694	124
6º	0.9694	88
7º	0.9694	32
8º	0.9680	159
9º	0.9680	121
10º	0.9680	110

Com isso decidi treinar a rede neural com 106 neurônios na camada oculta.

O próximo passo foi converter a rede neural para C utilizando o Emlearn. Aqui eu tive um pouco de dificuldade, pois por algum motivo que não consegui determinar, importar o emlearn em um arquivo que está importando o sklearn causa um erro. Portanto, a solução foi criar um outro arquivo que lê a rede neural treinada previamente através de um arquivo temporário criado com joblib.dump.

Isso permitiu ter 1 arquivo com emlearn e outro com o sklearn, e assim resolveu o conflito entre os pacotes.

Código do terminal

O terceiro ponto foi a criação de um terminal em python para enviar os dados. Para isso foi utilizado o pyserial conforme os exemplos de aula. Já que os dados do dataset são inteiros, foi possível otimizar o payload fazendo com que cada atributo corresponda a um byte da string, e cada linha seja enviada por inteiro para o microcontrolador.

O código também aguarda a resposta via serial no formato “ok**X**”, onde **X** é a resposta do modelo que varia de 0-9. O terminal então salva o retorno junto com a resposta de referência e após receber todas as linhas calcula a taxa de acerto.

Código do microcontrolador

Por fim, o código do microcontrolador foi configurado utilizando a UART conectada à porta USB. O código em si é simples, sendo somente um pooling no buffer de recepção que quando há 64 bytes o código lê, converte para float (por causa da implementação do emlearn), calcula a saída e retorna via serial. O laço while principal pode ser observado abaixo:

```c
  while (1)
  {
    if (HAL_UART_Receive(&huart2, (uint8_t *)rx_buff, 64, 1000) == HAL_OK)
    {
      HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

      float input[64];
      for (int i = 0; i < 64; i++)
      {
        input[i] = (float)rx_buff[i];
      }

      int resultado = rede_neural_predict(input, 64);
      sprintf(tx_buff, "ok%d", resultado);
      HAL_UART_Transmit(&huart2, (uint8_t *)tx_buff, 3, 1000);
    }
    /* USER CODE END WHILE */      You, 2 days ago • commit inicial com rede neu

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Resultados:

Após determinar que o melhor número de neurônios é 106, o modelo foi treinado novamente utilizando o conjunto de treino+validação e obteve 96,94% de taxa de acerto.

Com o modelo no microcontrolador o terminal enviou as 360 amostras do conjunto de testes em 5 segundos e também obteve uma taxa de acerto de 96,94%.

O artigo Luca (2020) atingiu 92% utilizando uma rede MLP com a função de ativação ReLU, que é a mesma utilizada nesse trabalho.

Referências:

Parisi, Luca. (2020). m-arcsinh: An Efficient and Reliable Function for SVM and MLP in scikit-learn. 10.48550/arXiv.2009.07530.