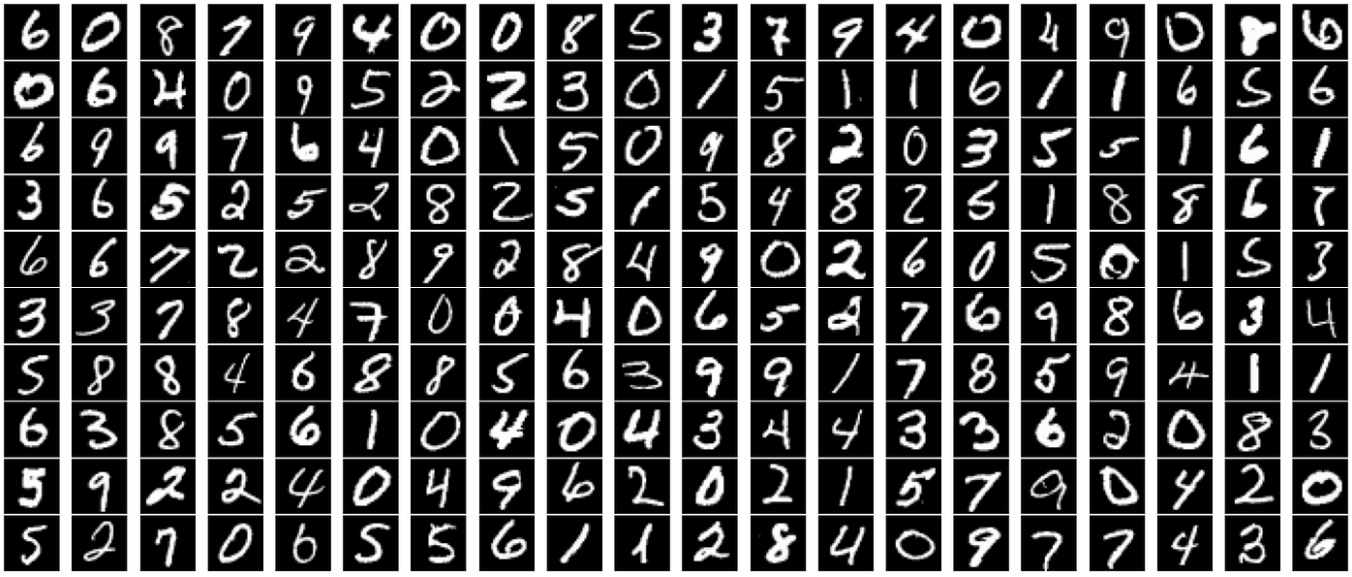
Classificador de Rede Neuronal em RISC-V

Este documento descreve os projetos da disciplina de Introdução à Arquitetura de Computadores 2024/2025 (LEIC-A e LEIC-T).

1 Introdução

Nestes três projetos, que podem ser vistos como as três fases de um projeto mais longo, irão entrar no mundo da linguagem Assembly RISC-V, explorando na prática conceitos fundamentais como o funcionamento de baixo nível do processador, as convenções de chamada de funções e, até, o projeto de um pequeno processador de 8 bits, dedicado a acelerar algumas das operações de uma rede neuronal.

A vossa missão? Construir, em Assembly, um classificador de dígitos manuscritos com base em algoritmos modernos de Inteligência Artificial! O objetivo é simular uma pequena rede neuronal, utilizando parâmetros de treino (pesos) e dados de entrada previamente definidos, com vista ao reconhecimento de dígitos de 0 a 9 a partir de imagens manuscritas, como as que se mostram na imagem seguinte.



É uma excelente oportunidade para ligarem a teoria à prática e perceberem como algoritmos de inteligência artificial podem ser implementados de forma eficiente, diretamente sobre a interface que o processador expõe ao software – isto é, o Assembly.

Para começarem com o pé direito, recomendamos que assistam ao vídeo abaixo – fornece um bom enquadramento sobre o que irão implementar.

[But what is a neural network? | Deep learning chapter 1]

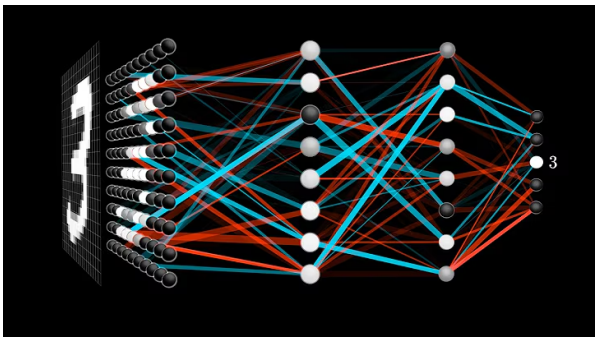


Figura 1: Representação visual da rede neuronal (imagem copiada do vídeo).

Projeto 1: Primeiros módulos

Neste primeiro projeto, cada grupo deve resolver os três problemas de aritmética inteira descritos de seguida.

Cada problema deve ser resolvido como um programa autónomo, pelo que a entrega consistirá em três ficheiros distintos.

1a. Valor absoluto de um inteiro (abs)

Este primeiro passo serve apenas para "aquecer". A função **abs** é uma rotina que calcula o **valor absoluto de um número inteiro** armazenado em memória:

- Carrega o valor apontado por um ponteiro
- Verifica se é negativo
- Se for, armazena o seu simétrico (positivo) na mesma posição de memória.

Argumentos

- `a0 (int *)`: Ponteiro para um único inteiro em memória.

Valores de Retorno

- Nenhum (a função modifica o valor diretamente na memória apontada).

Exceção

- Nenhuma.

1b. Rectified Linear Unit (relu)

A ReLU é uma função de ativação amplamente utilizada em redes neuronais, particularmente em modelos de aprendizagem profunda. Para cada inteiro num vetor de entrada, a função mantém o valor se for positivo; caso contrário, retorna altera-o para zero.

- ➢ **Argumentos**
  - `a0` (int *): Ponteiro para o início do vetor de inteiros.
  - `a1` (int): Número de inteiros no vetor.
- ➢ **Valores de Retorno**
  - Nenhum (a função modifica os valores diretamente no vetor)
- ➢ **Exceção**
  Se a entrada estiver malformada da forma indicada a seguir, deve colocar-se o código de erro apropriado em `a0` e saltar imediatamente para o final do programa com `j exit_with_error`.
  - Código 36 - O comprimento do vetor é menor que 1.

1c. Maior elemento de um vetor (argmax)

Esta função recebe um vetor de inteiros e retorna o **índice do maior elemento**. Se existirem múltiplos elementos com o mesmo valor máximo, deve ser devolvido o menor índice entre eles. Por exemplo, para o vetor de inteiros `[-6, -1, 6, 1]`, o resultado deve ser `2` (índice de `6`). Para o vetor `[6, 1, 6, 1]`, o resultado deve ser `0`, pois o valor `6` aparece pela primeira vez nessa posição.

- **Argumentos**
  - `a0` (int *): Ponteiro para o início do vetor de inteiros.
  - `a1` (int): Número de inteiros no vetor.
- **Valor de Retorno**
  - `a0` (int): Índice do maior elemento do vetor (em caso de empate, o menor índice).
- **Exceção**

  Tal como no exercício anterior, coloque o código de erro correspondente em `a0` e execute `j exit_with_error` para sair do programa.
  - Código 36 – O comprimento do vetor é inferior a 1.

Projeto 2: Compor a Rede Neuronal

Neste projeto iremos implementar mais algumas funções de processamento matricial, reaproveitar a lógica desenvolvida no primeiro projeto (agora organizada em três funções) e, finalmente, integrar todas estas funções para construir a nossa rede neuronal.

A rede neuronal a implementar é do tipo *feedforward* com uma única camada. Todos os pesos e *bias* são fornecidos. Não é necessário realizar o treino da rede.

2a. Função DotProduct (produto escalar)

Esta função recebe dois vetores de inteiros com o mesmo tamanho e devolve o respetivo produto escalar. Isto é, multiplica os elementos correspondentes entre si e retorna a soma de todos os produtos.

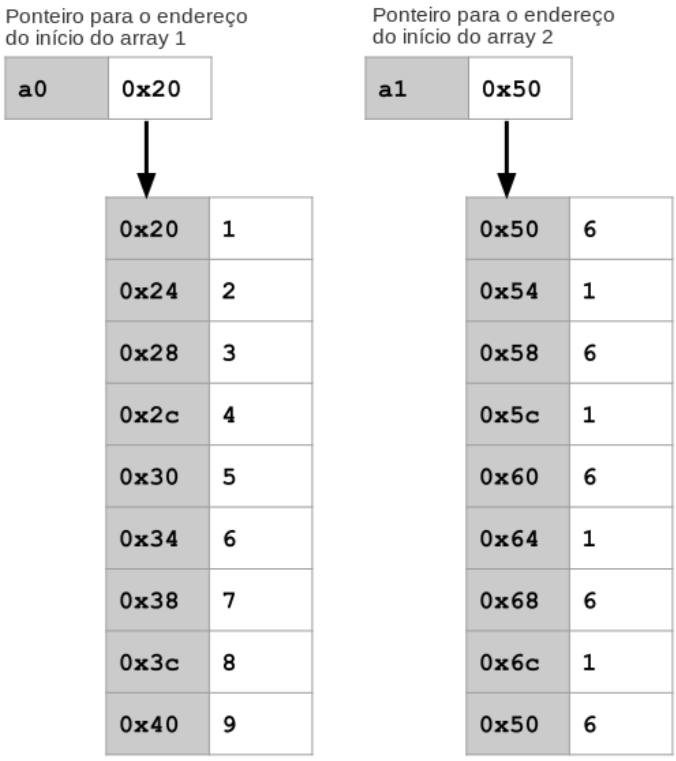


Figura 2: Exemplo da função `DotProduct`.

Por exemplo, se forem passados os dois vetores acima (Figura 2) à função `dotProduct` a função devolverá chegar ao seguinte resultado:

$$(1\times6) + (2\times1) + (3\times6) + (4\times1) + (5\times6) + (6\times1) + (7\times6) + (8\times1) + (9\times6) = \mathbf{170}.$$

Argumentos:

- `a0` (int *): Ponteiro para o início do primeiro vetor.
- `a1` (int *): Ponteiro para o início do segundo vetor.
- `a2` (int): Número de elementos de cada vetor.

Valores de Retorno:

- `a0` (int): Resultado do produto escalar entre os dois vetores.

Se a entrada estiver malformada da forma abaixo, coloque o código de erro apropriado em `a0` e execute `j exit_with_error` para terminar o programa:

Códigos de Retorno:

- **36**: O número de elementos a utilizar é menor que 1.

2b. MatMul (multiplicação de matrizes)

Esta função recebe duas matrizes inteiras, A (de dimensão $n \times m$) e B (de dimensão $m \times k$), e gera uma matriz de inteiros C (de dimensão $n \times k$). Para calcular o valor da entrada na linha i e coluna j de C, deve calcular-se o produto escalar entre a i-ésima linha de A e a j-ésima coluna de B.

- ➢ **Argumentos**
  - `a0` (int *): Ponteiro para o início da matriz A (armazenada como vetor de inteiros em ordem de linha).
  - `a1` (int *): Ponteiro para o início da matriz B (armazenada como vetor de inteiros em ordem de linha).
  - `a2` (int): Número de linhas (altura) da matriz A.
  - `a3` (int): Número de colunas (largura) da matriz A.
  - `a4` (int): Número de linhas (altura) da matriz B.
  - `a5` (int): Número de colunas (largura) da matriz B.
  - `a6` (int *): Ponteiro para o início da matriz C, que se assume ter a dimensão esperada.
- ➢ **Valores de Retorno**
  - Nenhum
- ➢ **Exceções**
  Se a entrada estiver malformada de uma das formas seguintes, coloque o código de erro apropriado em `a0` e execute `j exit_with_error` para terminar o programa.
  - Código de Retorno 38 - A altura ou a largura de qualquer uma das matrizes é inferior a 1.

- Código de Retorno 38 - O número de colunas da matriz A não é igual ao número de linhas da matriz B.

2c. Compor a rede neuronal

Lembrem-se: estamos a construir uma rede neuronal simples, passo a passo. Nesta tarefa, irão utilizar as funções desenvolvidas nas tarefas anteriores para completar a rotina de classificação, denominada **`classify`**.

A rotina **`classify`** deve implementar os seguintes passos:

1. Aplicar a função `matmul` para efetuar a multiplicação da matriz de pesos da primeira camada pela matriz de entrada da rede, obtendo a saída da primeira camada.
    - Esta operação corresponde à transformação linear inicial da rede. Serve para projetar os dados de entrada (por exemplo, os pixéis de uma imagem) para um novo espaço de características, onde relações mais relevantes para a tarefa de classificação podem ser modeladas. A matriz de pesos representa os parâmetros aprendidos durante o treino.
2. Aplicar a função `relu` sobre o resultado da operação anterior.
    - A função `relu` aplica uma ativação não-linear elemento a elemento, essencial para que a rede consiga modelar relações complexas nos dados.
3. Aplicar novamente a função `matmul` para multiplicar a matriz de pesos da segunda camada pelo resultado da relu.
    - Esta segunda multiplicação linear produz o resultado das ativações, um para cada classe possível. Estas ativações indicam o grau de compatibilidade da entrada com cada uma das classes da rede.

4. Aplicar a função `argmax` sobre a saída da matmul para determinar o índice da classe com o maior valor, assegurando que esse valor é colocado no registo apropriado antes de retornar da função.
    - Este último passo identifica qual das classes é a mais provável, de acordo com os valores de ativação obtidos. Esta é a decisão final da rede — a classe prevista para a entrada.

Posteriormente publicaremos uma extensão a este enunciado com pormenores sobre como concretizar os passos acima.

Projeto 3: Uma máquina de estados para acelerar operações da rede neuronal

Neste projeto, construirão um circuito sequencial, mais precisamente um processador de 8 bits que será especializado em algumas operações muito importantes para a vossa rede neuronal.

Esse processador será uma versão muito simplificada daquilo que hoje em dia se chama uma *tensor processing unit (TPU)*. Ela re-implementará a lógica das funções `abs` e `relu` do 1º projeto. No entanto, como essa implementação é feita em *hardware*, será mais otimizada. Portanto, este processador será aquilo que designamos um *acelerador*.

Posteriormente publicaremos uma extensão a este enunciado com pormenores sobre o 3º projeto.

4 Considerações práticas

O projeto deve ser desenvolvido no simulador **Ripes** (1º e 2º projetos) e ferramenta **Logisim** (3º projeto), que serão apresentados nas aulas laboratoriais. O processador simulado em Ripes deve ser um **RISC-V de 32 bits**, de acordo com a arquitetura estudada nas aulas teóricas e laboratoriais.

No caso dos 1º e 2º projetos, forneceremos os ficheiros de código iniciais (através do Fénix). As soluções devem **obrigatoriamente** ser construídas sobre esses ficheiros, respeitando a estrutura fornecida inicialmente.

5 Entregas e avaliação

Os projetos são submetidos no fénix, com os seguintes prazos:

- Projeto 1: 23h59 de 9/maio
- Projeto 2: 23h59 de 30/maio
- Projeto 3: 23h59 de 16/junho

Nota mínima de 9,0.

O peso de cada projeto na nota final pode ser consultado nos métodos de avaliação no fenix.

Embora o projeto seja desenvolvido em grupos de 3 estudantes, após as 3 submissões haverá uma discussão cujo objetivo é aferir se cada membro do grupo contribuiu e domina as soluções que foram submetidas pelo grupo. A discussão inclui um momento em que se pede a cada membro do grupo que faça modificações menores às soluções submetidas pelo grupo (no fundo, um pequeno teste prático).

Usaremos software de deteção de cópias. Casos de plágio seguem os procedimentos do regulamento de avaliação do IST.