embarca tech | Residência Tecnológica em Sistemas Embarcados

Fundamentos de Aprendizagem de Máquina para Sistemas Embarcados

Unidade 6 | Capítulo 1

Prof. Otávio Alcântara de Lima Jr.

Executores: INSTITUTO FEDERAL Piauí; INSTITUTO FEDERAL Rio Grande do Norte; INSTITUTO FEDERAL Maranhão; INSTITUTO FEDERAL Ceará; cepedi; HARDWARE.BR

Coordenação: Softex

Iniciativa: GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Sumário

Unidade 6

Capítulo 1

1. Introdução

Olá, estudantes!

Bem-vindos ao nosso material de apoio sobre "Fundamentos de Aprendizagem de Máquinas para Sistemas Embarcados". Este material é uma extensão do Capítulo 1 da Unidade 6 e foi desenvolvido para complementar seu aprendizado, oferecendo explicações adicionais e exemplos práticos.

A Aprendizagem de Máquinas é uma área da Inteligência Artificial focada em criar programas que aprendem progressivamente a resolver tarefas a partir de exemplos. Imagine, por exemplo, a tarefa de prever o valor de venda de imóveis. Um corretor de imóveis experiente é capaz de estimar o preço de uma casa com base em seu conhecimento. Agora, pense em um site de imóveis com dados sobre milhares de vendas, incluindo características detalhadas como metragem, localização e proximidade de transporte público. Esse grande conjunto de dados fornece uma visão ainda mais ampla do que a de um único especialista. Com a Aprendizagem de Máquinas, podemos criar um modelo que usa esses dados para fazer previsões de preço, muitas vezes com precisão superior à dos especialistas.

Na programação tradicional, programamos regras específicas para processar dados e produzir resultados. Em Aprendizagem de Máquinas, os dados e as respostas conhecidas são usados para criar essas regras, que o modelo aprende a partir dos exemplos. Assim, o modelo é capaz de gerar respostas precisas para novos dados. Essa técnica tem sido aplicada em diversas áreas – saúde, automação, robótica, indústria, entretenimento – e já faz parte do nosso dia a dia, desde aplicativos de recomendação até sistemas de segurança.

Inicialmente, os modelos de Aprendizagem de Máquinas eram muito

complexos para serem executados em sistemas embarcados, que são dispositivos com restrições de memória, processamento e energia. No entanto, novas técnicas surgiram para otimizar esses modelos, permitindo que rodem até mesmo em microcontroladores, como os que usamos neste curso. Isso possibilita que dados sejam processados diretamente onde são coletados, reduzindo a latência e a necessidade de comunicação constante com a nuvem. Para um desenvolvedor de sistemas embarcados, é essencial entender esses conceitos e conhecer as ferramentas para implementar algoritmos de Aprendizagem de Máquinas em dispositivos de baixo consumo.

A seguir, vamos ver como este capítulo está estruturado. Começamos apresentando um fluxo de trabalho para desenvolver e implementar um modelo de Aprendizagem de Máquinas em um microcontrolador, detalhando cada etapa para você entender o processo completo. Em seguida, exploramos a linguagem Python, fundamental para o desenvolvimento desses modelos, pois muitas das ferramentas são baseadas nessa linguagem. Na sequência, introduzimos a biblioteca Keras e mostramos como criar modelos simples para resolver problemas de classificação, regressão e processamento de imagens. Finalmente, fechamos com um resumo das informações e orientações sobre os próximos passos do aprendizado.

2. Fluxo de Projeto de Aprendizagem de Máquinas para Sistemas Embarcados

Nesta seção, apresentamos um fluxo simplificado para desenvolver projetos de Aprendizagem de Máquinas em sistemas embarcados, com ênfase nas ferramentas disponíveis na plataforma TensorFlow. Embora existam diversos frameworks para aplicar Aprendizagem de Máquinas em microcontroladores, nosso foco será o TensorFlow Lite para Microcontroladores [1], que é amplamente utilizado para implantar modelos leves em dispositivos com poucos recursos.

Quando falamos especificamente do microcontrolador RP2040 [2], que está na base da placa BitDogLab [3], o TensorFlow Lite Micro [4] surge como uma opção prática para execução de modelos de Aprendizagem de Máquinas. Além disso, exploraremos o uso da biblioteca Edge Impulse [5], que oferece um ambiente integrado para criação e implementação

de modelos em dispositivos embarcados.

Para simplificar, nos referiremos à área de Aprendizagem de Máquinas em sistemas embarcados como TinyML. TinyML representa a interseção entre três campos: sistemas embarcados, Aprendizagem de Máquinas e aplicações práticas. Dominar essa área exige uma reflexão sobre cada um desses aspectos, pois o verdadeiro potencial do TinyML só pode ser alcançado quando compreendemos as limitações e possibilidades específicas desses sistemas.

Algumas perguntas são essenciais para uma compreensão aprofundada do TinyML e guiarão nosso estudo nesta unidade:

Como a Aprendizagem de Máquinas permite criar aplicações TinyML inovadoras?

TinyML possibilita levar a inteligência artificial para dispositivos compactos e baratos, expandindo o uso da tecnologia para áreas como monitoramento remoto, reconhecimento de gestos e controle inteligente de dispositivos.

Quais são os principais desafios ao executar modelos de Aprendizagem de Máquinas em dispositivos de recursos limitados?

A execução de modelos em sistemas com pouca memória e baixo processamento traz desafios específicos, como a necessidade de otimização de modelos e o gerenciamento do consumo de energia, exigindo soluções que minimizem o uso de recursos.

Que novas aplicações podemos desenvolver hoje com TinyML?

Com TinyML, podemos criar soluções que antes exigiam processamento centralizado ou em nuvem. Agora, dispositivos autônomos podem processar dados localmente, como sensores de saúde que identificam padrões de movimento em tempo real ou câmeras de segurança que detectam eventos automaticamente

Ao longo desta unidade, você será convidado a refletir sobre essas questões, visando construir o entendimento e as habilidades necessárias para aplicar TinyML em seus próprios projetos.

A Figura 1 ilustra os principais elementos de uma aplicação TinyML. Os sensores são componentes essenciais, pois permitem que a aplicação capture informações do mundo físico. Existem vários tipos de sensores utilizados em aplicações TinyML, incluindo sensores acústicos, de movimento e de imagem, entre outros. Cada um deles desempenha um papel crucial na coleta de dados que servirão de base para a inferência.

Uma aplicação TinyML geralmente é executada em um dispositivo embarcado simples, como um microcontrolador ou um dispositivo edge. Esse dispositivo não só realiza as funções típicas de um sistema embarcado, mas também executa inferências de modelos de Aprendizagem de Máquinas com os dados obtidos dos sensores. Em muitos casos, o dispositivo embarcado faz parte de um sistema IoT, permitindo que os resultados sejam reportados para uma aplicação na nuvem. Isso possibilita a análise e o monitoramento remoto, ampliando o alcance e a utilidade das soluções TinyML.

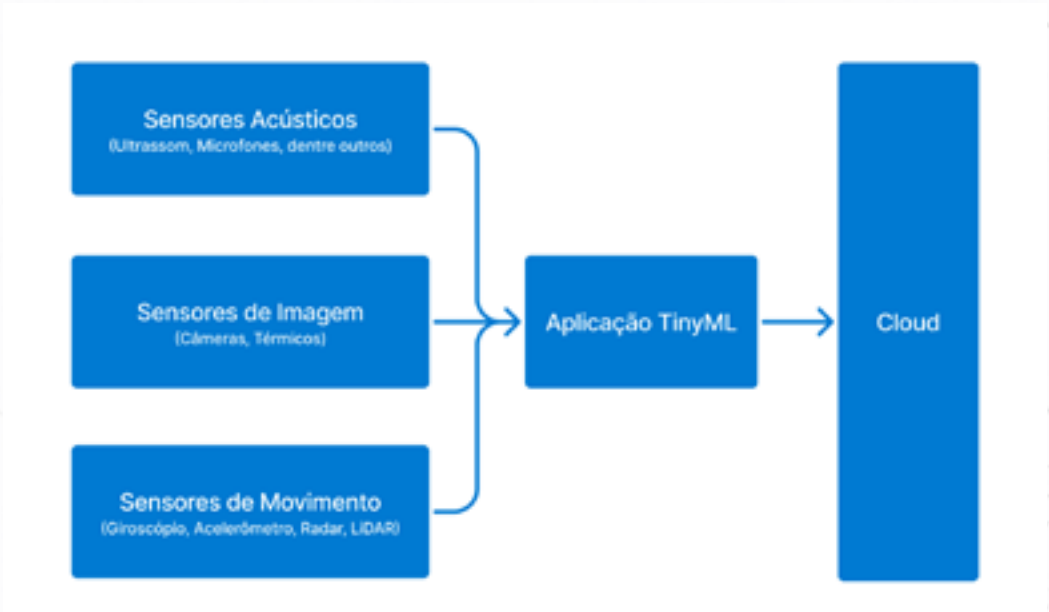


Figura 1: diagrama de blocos de uma aplicação TinyML.
Fonte: Imagem do autor.

A partir de agora, apresentaremos um fluxo simplificado de projeto TinyML, que vai desde a coleta de dados até o funcionamento do sistema embarcado em campo. A Figura 2 ilustra esse fluxo simplificado.

As primeiras etapas fazem parte do que chamamos de engenharia de dados. A fase inicial é a coleta de dados, na qual especialistas no domínio do problema aplicam métodos estatísticos para reunir dados empíricos sobre a tarefa que o modelo deverá aprender. Esses dados formam a base do treinamento, e sua qualidade é essencial para que o modelo aprenda de forma eficaz.

Depois da coleta, os dados passam por um pré-processamento. Nessa fase, são adicionados rótulos para as saídas, e os dados são inspecionados, limpos e validados, garantindo consistência e removendo informações irrelevantes ou incorretas. Em casos em que há limitação de dados ou o custo de coleta é alto, a técnica de Data Augmentation [6] pode ser utilizada. Com ela, é possível gerar novos dados a partir dos existentes, ampliando a base de dados e enriquecendo o aprendizado do modelo.

Essas etapas de engenharia de dados são fundamentais para preparar os dados de forma que o modelo possa identificar e aprender padrões relevantes, aplicando-os posteriormente em novas situações.

Figura 2: fluxo simplificado de projeto TinyML.
Fonte: Imagem do autor.

A fase seguinte é conhecida como Engenharia de Modelo e inclui o projeto e o treinamento do modelo. Na etapa de projeto, os especialistas decidem qual tipo de modelo é mais **adequado** para resolver o problema em questão. Após essa definição, o modelo é submetido ao treinamento, onde ele aprende a partir dos dados fornecidos.

É comum que essas etapas sejam repetidas algumas vezes. Com base

nos resultados iniciais do treinamento, os especialistas podem ajustar o modelo, alterando o número de camadas ou tipos de camadas utilizadas para melhorar o desempenho. Esse processo de refinamento é essencial para garantir que o modelo seja eficiente e preciso ao resolver o problema proposto.

Após o treinamento, passamos para a fase de Implantação do Modelo. Essa fase envolve três etapas: otimização, conversão e implantação.

Na etapa de otimização, o modelo é ajustado para reduzir o uso de memória. Uma das técnicas mais comuns é a **quantização dos pesos**, que consiste em utilizar representações numéricas mais simples para armazenar os pesos do modelo. Isso reduz a precisão ligeiramente, mas também diminui o consumo de memória, tornando o modelo mais adequado para dispositivos com poucos recursos.

Em seguida, o modelo é convertido para o formato compatível com a plataforma embarcada, utilizando ferramentas como TensorFlow Lite e TensorFlow Lite Micro. Após a conversão, o modelo está pronto para ser implantado diretamente no microcontrolador, tornando-o capaz de realizar inferências no próprio dispositivo.

A fase final é a Inferência, onde o modelo implantado opera em campo com dados reais. Nesta etapa, é importante implementar mecanismos de monitoramento da performance do sistema TinyML para que o modelo possa ser continuamente melhorado. Esse monitoramento permite avaliar se o modelo está funcionando conforme esperado e identificar possíveis ajustes necessários para otimizar sua eficiência e precisão.

Nesta seção, apresentamos um fluxo simplificado para o desenvolvimento de projetos em TinyML. É importante manter em mente cada fase e etapa descrita, pois elas refletem o passo a passo de um projeto real. Durante esse fluxo, é comum realizar várias iterações para ajustar e melhorar o modelo, um processo que você compreenderá melhor ao colocá-lo em prática.

Na próxima seção, apresentaremos um tutorial introdutório de programação em Python. Nele, você aprenderá a utilizar as bibliotecas TensorFlow e Keras para criar modelos de Aprendizagem de Máquinas,

dando o primeiro passo para desenvolver suas próprias aplicações TinyML.

3. Um Breve Tutorial de Python

Python é a **principal** linguagem usada em ciência de dados, Aprendizagem de Máquinas e áreas relacionadas. Por isso, é **essencial** desenvolvermos um conhecimento **prático** sobre como criar programas básicos em Python. Nesta seção, apresentaremos um breve tutorial de programação em Python e recomendamos que o leitor explore outras fontes, como [7, 8], para aprofundar-se ainda mais.

A linguagem Python foi criada em 1990 pelo matemático holandês Guido van Rossum. Python é uma linguagem interpretada, orientada a objetos e de tipagem dinâmica. Em pouco tempo, conquistou um lugar de destaque e se tornou uma das linguagens de programação mais influentes e amplamente utilizadas em diversas áreas.

Nesta seção, apresentaremos um breve tutorial de programação em Python, que ajudará você a entender melhor os exemplos de código usados na parte de Aprendizagem de Máquinas. Para aproveitar ao máximo, é importante que você experimente e repita os exemplos, o que facilitará o entendimento dos conceitos.

Começaremos explorando as classes embutidas da linguagem para trabalhar com diferentes tipos de dados. Em seguida, abordaremos expressões e operadores, fundamentais para realizar operações e manipulações em Python. Também veremos como utilizar as estruturas de controle e repetição, que permitem tomar decisões e executar ações repetidas. Por fim, aprenderemos a criar e usar funções, um recurso essencial para organizar e reutilizar código de forma eficiente.

3.1 Criando e usando objetos das classes embutidas

Em Python, as classes formam a base de todos os tipos de dados. Vamos aprender a criar objetos, utilizar métodos e explorar as classes embutidas que nos permitem trabalhar com os tipos de dados fundamentais, como

inteiros, números de ponto flutuante, listas, tuplas, strings e dicionários.

Para começar, veremos como criar objetos em Python. Uma instância de um objeto pode ser criada utilizando um construtor específico da classe ou diretamente pela atribuição de um valor literal. A Figura 3 mostra um exemplo de como instanciar objetos dessa forma.

```
1   a = int(4.5)
2   b = 4.5
3   c = "Hello"
4   d = ['maçã','banana','laranja']
5   e = d[1]
6   f = int(0xff563)
7   g = bool(1)
8   h = (2,3,6)
9   i = {'Brasil':'real', 'EUA':'dolar', 'Japan':'yen'}
10  j = i['Brasil']
11
12  print(a,b,c,d,e,f,g,h,i,j)
```

Figura 3: exemplo de criação de objetos das classes embutidas.
Fonte: imagem do autor.

Na primeira linha do código, vemos como instanciar um objeto inteiro usando o construtor da classe int. Passamos como parâmetro um número em ponto flutuante, e o construtor converte esse valor, removendo a parte decimal. A variável a passa a referenciar uma região de memória que contém um objeto do tipo int com o valor 4.

Em seguida, criamos um objeto do tipo ponto flutuante, atribuindo o valor diretamente à variável b, que agora armazena o número 4.5. A variável c aponta para um objeto do tipo str (string), que foi construído por meio de uma atribuição direta do texto “Hello”.

A variável d é uma lista contendo nomes de frutas, e cada elemento da lista é uma string. Listas são coleções de elementos que podem ser acessados por índice. Ao acessarmos o índice 1 da lista d, obtemos o segundo elemento, ‘banana’, que é atribuído à variável e. Assim, e passa a apontar para o valor ‘banana’.

Na próxima linha, a variável f armazena um número inteiro construído a partir de um valor em hexadecimal (0xff563). O Python converte automaticamente esse valor hexadecimal para seu equivalente em decimal e o armazena em f.

A variável g é um valor booleano, que recebe True como resultado da função bool(1). Em Python, 1 representa verdadeiro, então g armazena True.

A variável h é uma tupla de inteiros (2, 3, 6). Diferente de listas, tuplas são imutáveis, ou seja, os valores armazenados nelas não podem ser alterados após a criação.

A variável i é um dicionário, uma estrutura que mapeia chaves para valores. No exemplo, i armazena pares de chave-valor onde 'Brasil' está associado a 'real', 'EUA' a 'dolar', e 'Japan' a 'yen'. Finalmente, a variável j armazena o valor 'real', acessado no dicionário i por meio da chave 'Brasil'.

Vale destacar que, em Python, ao contrário de linguagens como C e C++, não precisamos declarar o tipo das variáveis. O interpretador identifica automaticamente o tipo de dado ao qual uma variável se refere, facilitando o processo de codificação e aumentando a flexibilidade do código.

3.2 Expressões e operadores

```
1   a = 5
2   b = 3
3   c = a**b
4   c += b
5   c -= b
6   c = a + b
7   c = a - b
8   c = a * b
9   c = a / b
10  c = a // b
11  c = a % 10
12  print(c)
```

Vamos analisar alguns exemplos de código para mostrar como trabalhamos com expressões e operadores na linguagem Python. Na Figura 4 apresentamos alguns exemplos de manipulação de variáveis inteiras com operadores aritméticos.

Figura 4: exemplo de uso de operadores aritméticos.
Fonte: imagem do autor.

Após inicializar as variáveis a e b com os valores 5 e 3, iniciamos uma série de manipulações usando operadores aritméticos em Python.

Primeiro, aplicamos o operador de exponenciação **, calculando a elevado a b, ou seja, 5 elevado a 3, e armazenando o resultado em c. Em seguida, utilizamos os operadores += e -=, que permitem realizar operações de adição e subtração de forma simplificada. Especificamente, c += b é equivalente a c = c + b, enquanto c -= b representa c = c - b.

Nas linhas seguintes, apresentamos operações básicas de aritmética: soma (c = a + b), subtração (c = a - b), multiplicação (c = a * b), e divisão (c = a / b). A divisão com o operador / resulta em um valor de ponto flutuante, mesmo que ambos os operandos sejam inteiros.

Na linha 10, introduzimos o operador de divisão inteira //, que retorna apenas a parte inteira do resultado, descartando o valor após a vírgula. Esse operador é útil quando precisamos de um resultado inteiro e não queremos lidar com decimais.

Por fim, usamos o operador de módulo % para calcular o resto da divisão de a por 10. Isso nos dá o valor que sobra quando a é dividido por 10, uma operação útil em várias situações, como determinar se um número é par ou ímpar ou extrair dígitos específicos de um número.

Esses são alguns dos principais operadores aritméticos em Python, e entender como usá-los é fundamental para manipular valores numéricos de maneira eficiente no código.

Outro conjunto de operadores importantes no dia a dia da programação em Python são os operadores booleanos bitwise, que permitem realizar operações diretamente nos bits de um número. O código abaixo demonstra o uso desses operadores.

Primeiro, temos os operadores de deslocamento à direita (>>) e à esquerda (<<). Na linha 4, a é deslocado um bit para a esquerda, o que equivale a multiplicá-lo por 2. Na linha 5, b é deslocado um bit para a direita, o que equivale a dividir b por 2. Esses operadores são úteis para manipulações rápidas, especialmente quando lidamos com operações de baixo nível, como otimizações e algoritmos específicos.

Em seguida, vemos os operadores E (&), OU (|) e XOR (^) bit-a-bit. Esses operadores aplicam a operação lógica a cada bit dos números. Por exemplo, o operador E (&) retorna 1 apenas nos bits onde ambos os operandos têm 1, enquanto o operador OU (|) retorna 1 em qualquer bit onde pelo menos um dos operandos tenha 1. O operador XOR (^) retorna 1 apenas nos bits onde os operandos diferem entre si.

```
1   a = 12
2   b = 6
3
4   c = a << 1
5   c = b >> 1
6   c = a & b
7   c = a | b
8   c = a ^ b
9   print(c)
```

Esses operadores são frequentemente usados em operações que envolvem manipulação direta de bits, como em algoritmos de criptografia, compressão de dados, controle de dispositivos de hardware e otimização de memória. Execute o exemplo para observar como as variáveis são manipuladas e entender melhor o funcionamento desses operadores bitwise.

Figura 5: uso dos operadores bitwise.
Fonte: imagem do autor.

Quando escrevemos um programa, é comum precisarmos fazer comparações entre variáveis ou valores constantes. Para isso, usamos operadores de comparação e operadores lógicos específicos. O exemplo abaixo ilustra o uso desses operadores.

Nas primeiras linhas, vemos operadores de comparação básicos: igualdade (==), diferença (!=), maior que (>), maior ou igual a (>=), menor que (<) e menor ou igual a (<=). Esses operadores permitem verificar relações entre valores, retornando True ou False dependendo do resultado da comparação. Por exemplo, a == b verifica se a é igual a b, enquanto a != b verifica se a é diferente de b.

Além disso, temos o operador lógico de negação not, que inverte o valor booleano de uma expressão. Na linha 10, not a == b retorna True se a não for igual a b.

Nas últimas linhas, utilizamos os operadores lógicos and e or, que permitem combinar múltiplas condições. A expressão (a != b) and (a < 20)

retorna True apenas se ambas as condições forem verdadeiras (ou seja, a for diferente de b e a for menor que 20). Já a expressão (a != b) or (a > 10) retorna True se pelo menos uma das condições for verdadeira.

Esses operadores de comparação e lógicos são fundamentais para controlar o fluxo do programa, permitindo a execução de ações baseadas em condições específicas. Analise e experimente esse exemplo para solidificar sua compreensão sobre como essas comparações funcionam em Python.

```
1   a = 12
2   b = 6
3
4   c = a == b
5   c = a != b
6   c = a > b
7   c = a >= b
8   c = a < b
9   c = a <= b
10  c = not a == b
11  c = (a!=b) and (a < 20 )
12  c = (a!=b) or ( a > 10)
13  print(c)
```

Figura 6: operadores de comparação e lógicos.
Fonte: imagem do autor.

Por fim, vamos explorar alguns exemplos de operadores de comparação e pertencimento em Python. O código abaixo utiliza duas listas: a primeira (a) contém nomes de brinquedos, e a segunda (b) contém objetos domésticos.

Primeiro, temos um exemplo de sobrecarga de operadores com o operador +. Quando aplicado a duas listas, o operador + funciona como um operador de concatenação, unindo as listas a e b em uma única lista, que é atribuída à variável c.

Em seguida, usamos os operadores in e not in para verificar se um elemento está presente em uma lista. Na linha 5, verificamos se 'bola' está na lista a, e o resultado será True porque 'bola' é um dos elementos

de a. Na linha 6, verificamos se 'casa' não está na lista a, e o resultado será True porque 'casa' realmente não é um elemento de a.

Por fim, mostramos como verificar o tipo de uma variável usando os operadores is e is not. O operador is verifica se uma variável é de um tipo específico. Na linha 8, c = a is int verifica se a é um objeto do tipo int, enquanto na linha 9, c = a is not tuple verifica se a não é uma tupla. Esses operadores são úteis para checar o tipo de dados, garantindo que as variáveis estejam no formato esperado antes de realizar operações.

Esse conjunto de operadores facilita a verificação de pertencimento e o controle de tipos em Python, tornando o código mais legível e robusto.

```
1  a = ['bola', 'boneca', 'carro', 'xadrez']
2  b = ['panela','mesa','cadeira']
3
4  c = a + b
5  c = 'bola' in a
6  c = 'casa' not in a
7
8  c = a is int
9  c = a is not tuple
10
11
12 print(c)
```

Figura 7: exemplo de operadores de pertencimento e comparação.
Fonte: imagem do autor.

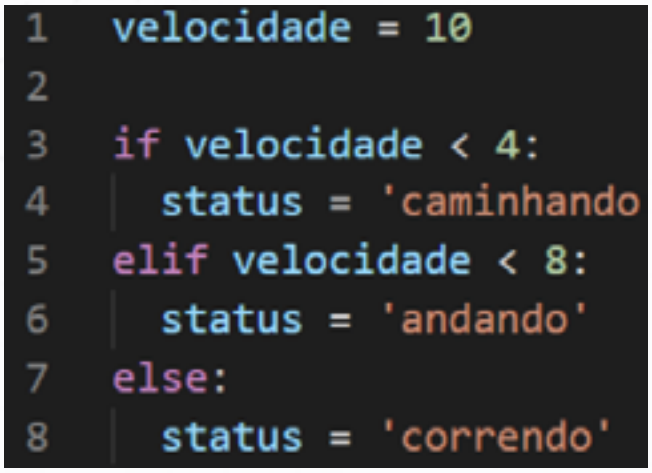
3.3 Controle de Fluxo

Vamos explorar exemplos simples de estruturas de controle e repetição em Python. Para programadores acostumados com linguagens como C e C++, alguns aspectos de Python podem chamar a atenção. Em C e C++, blocos de código são delimitados por chaves { }, enquanto em Python utilizamos a indentação (espaços ou tabulações) para definir esses blocos. Além disso, Python não exige ponto e vírgula ; no final de cada linha, o que torna o código mais limpo e legível.

Começaremos com a estrutura if-elif-else. O código abaixo apresenta um exemplo simples onde essa estrutura é usada para determinar a atividade de uma pessoa com base em uma variável que indica sua velocidade.

No exemplo, a variável velocidade é inicializada com o valor 10. A partir daí, o bloco if-elif-else é utilizado para verificar diferentes faixas de velocidade. Se velocidade for menor que 4, o status será definido como 'caminhando'. Se for menor que 8, o status será 'andando'. Para qualquer outro valor (neste caso, 10), o status será definido como 'correndo'.

Esse exemplo ilustra como o if-elif-else permite a criação de decisões condicionais de maneira clara e intuitiva em Python, utilizando apenas a indentação para organizar o código em blocos.



```
1  velocidade = 10
2
3  if velocidade < 4:
4      status = 'caminhando
5  elif velocidade < 8:
6      status = 'andando'
7  else:
8      status = 'correndo'
```

Figura 8: exemplo de if-else.
Fonte: imagem do autor.

Agora vamos explorar exemplos de laços em Python, começando com o laço while. O código abaixo utiliza um laço while para percorrer uma lista chamada data, que contém alguns caracteres. A variável j é inicializada como 0 e será usada como índice para acessar os elementos da lista.

No while, temos duas condições: j < len(data) e data[j] != 'x'. O laço continuará enquanto ambas as condições forem verdadeiras. Isso significa que ele irá percorrer os elementos da lista data enquanto j for menor que o comprimento da lista e o caractere atual não for igual a 'x'.

A cada iteração, j é incrementado em 1 (j += 1), o que faz o índice avançar para o próximo elemento da lista. Quando o caractere 'x' é encontrado ou j atinge o final da lista, o laço while para de executar.

Esse exemplo ilustra como usar o laço while para iterar sobre uma estrutura de dados até encontrar um valor específico ou até que uma condição de limite seja atingida.

```
1  data = ['a', 'b', 'd', 'x', 'k']
2  j = 0
3
4  while j<len(data) and data[j] != 'x':
5      j +=1
```

Figura 9: exemplo de while.
Fonte: imagem do autor.

Agora vamos ver um exemplo de laço for em Python, que utiliza um iterable (no caso, uma lista) para gerar os elementos acessados durante as iterações.

Neste código, temos uma lista chamada data contendo alguns caracteres. Usamos o laço for para percorrer cada elemento da lista, atribuindo cada valor à variável item em cada iteração.

Dentro do laço for, temos uma estrutura if que verifica se o item atual é igual a 'x'. Se essa condição for verdadeira, a instrução break é acionada, interrompendo o laço antes de alcançar o final da lista.

Esse exemplo demonstra como o laço for pode ser usado para percorrer todos os elementos de um iterable, como uma lista, e como a instrução break permite interromper o laço de forma antecipada quando uma condição específica é atendida.

```
1  data = ['a', 'b', 'd', 'x', 'k']
2
3  for item in data:
4      if item == 'x':
5          break
```

Figura 10: exemplo de laço for.
Fonte: imagem do autor.

3.4 Funções

Agora, vamos ver um exemplo simples de como definir e usar funções com parâmetros polimórficos em Python.

O código define uma função chamada count que recebe dois parâmetros: data e target. O parâmetro data é uma lista que será percorrida pela função, e target é o valor que queremos contar dentro dessa lista. O target tem um valor padrão de 'x', o que significa que, se não passarmos um segundo argumento ao chamar a função, ela contará automaticamente o número de ocorrências de 'x'.

Dentro da função, inicializamos uma variável n com o valor 0 para armazenar a contagem de ocorrências do target na lista. Em seguida, usamos um laço for para percorrer cada item na lista data. Se o item for igual ao target, incrementamos n em 1. Após o laço, a função retorna o valor de n, que representa o número de vezes que target foi encontrado em data.

Nas linhas seguintes, criamos duas listas: d, contendo os itens 'casa', 'mesa', e 'cadeira', e e, que inclui os caracteres 'a', 'x', 'x', 'b' e mais um 'x'.

As últimas duas linhas de código chamam a função count com as listas d e e. Na primeira chamada, o target é especificado como 'casa', então a função conta quantas vezes 'casa' aparece na lista d. Na segunda chamada, a função é usada sem especificar um target, então ela usa o valor padrão 'x' e conta as ocorrências de 'x' na lista e.

Este exemplo mostra como funções em Python podem usar parâmetros polimórficos e valores padrão, tornando-as mais flexíveis e reutilizáveis em diferentes contextos.

```python
def count(data, target='x'):
    n = 0
    for item in data:
        if item == target:
            n += 1
    return n

d = ['casa','mesa', 'cadeira']
e = ['a', 'x', 'x', 'b', 'x']

print(count(d,'casa'))
print(count(e))
```

Figura 11: exemplo de função com parâmetro polimórfico.
Fonte: imagem do autor.

4. Exemplos Simples com Keras

Após aprendermos mais sobre programação em Python, podemos nos aventurar em alguns exemplos simples usando TensorFlow e Keras. Keras é uma biblioteca voltada para o desenvolvimento de modelos de Aprendizagem de Máquinas e está integrada ao TensorFlow, o que a torna poderosa e acessível.

Existem muitas vantagens em utilizar Keras. Primeiramente, ela oferece uma interface de alto nível, o que significa que é modular, intuitiva e fácil de usar. Keras permite que você construa e treine redes neurais complexas com um mínimo de código, facilitando a criação de protótipos e o desenvolvimento de soluções de Aprendizagem de Máquinas. Além disso, Keras possui suporte para unidades de processamento gráfico (GPUs), o que acelera significativamente o treinamento de modelos, especialmente para redes neurais profundas que demandam maior poder de processamento.

Usando Keras, você poderá explorar uma ampla variedade de arquiteturas de redes neurais, como redes convolucionais (CNNs) e redes recorrentes (RNNs), permitindo que você experimente soluções para diferentes tipos de problemas de maneira prática e eficiente.

Nesta seção apresentaremos dois exemplos de modelos de aprendizagem de máquinas escritos com o Keras, que foram adaptados de [9].

Neste primeiro exemplo, utilizamos um conjunto de dados clássico com informações sobre diferentes tipos de flores. Primeiramente, importamos as bibliotecas necessárias: pandas e as classes Sequential e Dense do Keras. A biblioteca pandas é usada para manipulação de dados, facilitando o processamento do arquivo CSV que contém o conjunto de dados. As classes Sequential e Dense do Keras nos ajudam a construir e estruturar nosso modelo de rede neural.

Não se preocupe com os detalhes do funcionamento da rede neural neste momento; eles serão explicados no próximo capítulo. O objetivo aqui é que você se familiarize com a forma como o modelo é descrito em Keras. Ao criar um modelo com Keras, focamos principalmente na descrição da estrutura física do modelo (camadas) e nos parâmetros usados para o treinamento.

Usamos a função read_csv do pandas para carregar o conjunto de dados a partir de um endereço na web. Em seguida, os valores das colunas sepal_l, sepal_w, petal_l, e petal_w são extraídos e atribuídos à variável x, que será a entrada do modelo. A coluna species, que representa o tipo de flor, é processada usando pd.get_dummies para transformá-la em uma representação numérica, e o resultado é armazenado em y, que será a saída do modelo.

O dataset utilizado no exemplo é o famoso conjunto de dados Iris, frequentemente empregado em estudos de estatística, aprendizado de máquina e ciência de dados por seu tamanho pequeno e simplicidade. O dataset foi introduzido pelo estatístico e biólogo Ronald A. Fisher em 1936 e contém informações sobre três espécies de flores de íris: Setosa, Versicolor e Virginica. Ele é particularmente útil para demonstrar classificações multiclasse, pois inclui características medíveis de cada flor que permitem diferenciá-las.

Nas próximas linhas, descrevemos a estrutura das camadas do modelo de rede neural. A primeira camada contém 50 neurônios e recebe a entrada x. A função de ativação usada é a relu, que será explicada em detalhes mais adiante. O importante agora é entender como cada camada é adicionada

ao modelo. Uma segunda camada é inserida com 25 neurônios e também usa a função de ativação relu. A camada final é uma camada densa com um número de neurônios igual ao número de classes em y, e usa a função de ativação softmax, que permite que o modelo retorne a probabilidade de cada flor pertencer a uma das classes (ou espécies) de saída.

Em seguida, compilamos o modelo, especificando uma função de perda e um otimizador. Esses parâmetros são fundamentais para o processo de treinamento, pois guiam o ajuste dos pesos da rede neural para melhorar sua precisão.

Finalmente, o treinamento do modelo é feito com a função fit, que utiliza os dados x e y, permitindo ao modelo aprender a partir dos exemplos fornecidos. Execute o exemplo em seu computador e veja os resultados do processo de treinamento. No próximo capítulo, iremos nos aprofundar na teoria das redes neurais profundas, bem como no uso do Keras.

```
import pandas as pd
from keras import Sequential
from keras.layers import Dense



df = pd.read_csv("https://data.heatonresearch.com/data/t81-558/iris.csv", na_values=['NA','?'
x = df[['sepal_l','sepal_w','petal_l','petal_w']].values
dummies = pd.get_dummies(df['species'])
species = dummies.columns
y = dummies.values

model = Sequential()
model.add(Dense(50,input_dim=x.shape[1],activation='relu'))
model.add(Dense(25,activation='relu'))
model.add(Dense(y.shape[1],activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adam')
model.fit(x,y,verbose=2,epochs=100)
pred = model.predict(x)
```

Figura 12: exemplo de classificação com Keras.
Fonte: imagem do autor.

Este exemplo demonstra um problema de regressão usando um conjunto de dados com informações sobre carros antigos. Diferente do exemplo anterior, além de Keras e pandas, importamos também as bibliotecas numpy e sklearn, que serão utilizadas para calcular a métrica de avaliação do modelo.

Após importar as bibliotecas, usamos pandas para carregar o conjunto de dados a partir de um endereço na web. Em seguida, aplicamos algumas operações de pré-processamento. Esse conjunto de dados contém valores ausentes na coluna horsepower, e substituímos esses valores pela mediana dos valores existentes para evitar problemas durante o treinamento.

A variável x representa a entrada do modelo, composta pelas colunas cylinders, displacement, horsepower, weight, acceleration, year e origin. Essas características representam especificações do veículo que podem ajudar a prever o consumo de combustível. A saída do modelo, y, é a coluna mpg, que indica o consumo de combustível do carro em milhas por galão. Nosso objetivo é prever o valor de mpg com base nas características do veículo.

A estrutura do modelo é semelhante ao exemplo anterior. Definimos três camadas densas usando Keras. A última camada tem apenas um neurônio, o que é típico em problemas de regressão, já que estamos prevendo um único valor numérico (o consumo do veículo).

Usamos as funções compile e fit para configurar o modelo, especificando a função de perda (mean_squared_error) e o otimizador (adam), e para realizar o treinamento da rede.

Por fim, avaliamos a precisão do modelo usando a média dos erros quadrados (RMSE), que é uma métrica comum para avaliar modelos de regressão. Quanto menor o valor do RMSE, mais preciso o modelo é na previsão do consumo de combustível dos veículos.

```
from keras import Sequential
from keras.layers import Dense
import pandas as pd
import numpy as np
from sklearn import metrics

df = pd.read_csv("https://data.heatonresearch.com/data/t81-558/auto-mpg.csv", na_values=['NA','?'])
cars = df['name']
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())
x = df[['cylinders','displacement','horsepower','weight','acceleration','year','origin']].values
y = df['mpg'].values

model = Sequential()
model.add(Dense(25,input_dim=x.shape[1],activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['acc'])
model.fit(x,y,verbose=2,epochs=100)

pred = model.predict(x)
score = np.sqrt(metrics.mean_squared_error(pred,y))
print("Final score:", {score})
```

Figura 13: exemplo de problema de regressão com Keras.
Fonte: imagem do autor.

Para consolidar os dois exemplos, vamos apresentar uma tabela com as principais estruturas mencionadas.

pd.read_csv	Função da biblioteca pandas que lê um arquivo CSV e retorna um DataFrame com os dados. Permite carregar dados de arquivos locais ou da web.
df['mpg'].values	Atributo do pandas DataFrame que retorna os valores do DataFrame como uma matriz NumPy, facilitando a manipulação e integração com outras bibliotecas.
Sequential	Classe do Keras que define um modelo de rede neural sequencial, onde as camadas são empilhadas em sequência.
Dense	Classe do Keras que cria uma camada densa (ou totalmente conectada) em uma rede neural. Cada neurônio está conectado a todos os neurônios da camada anterior.
model.add	Método da classe Sequential usado para adicionar uma nova camada ao modelo.
model.compile	Método da classe Sequential que configura o modelo com uma função de perda, um otimizador e métricas, preparando-o para o treinamento.

model.fit	Método da classe Sequential que treina o modelo nos dados de entrada e saída especificados, ajustando os pesos do modelo ao longo de várias épocas.
model.predict	Método da classe Sequential usado para realizar previsões com base em novos dados de entrada após o treinamento do modelo.

5. Conclusão

Ao longo deste documento, exploramos os fundamentos da Aprendizagem de Máquinas aplicada a sistemas embarcados, com foco em conceitos, ferramentas e práticas essenciais para o desenvolvimento de soluções inteligentes em dispositivos com recursos limitados. Discutimos a importância do pré-processamento de dados, a construção e o treinamento de modelos de redes neurais com Keras e TensorFlow, além de abordarmos o uso do Python como uma linguagem poderosa e acessível para implementar esses modelos.

Este material é apenas o ponto de partida. Nos próximos capítulos, abordaremos a teoria e a prática das redes neurais profundas, para que você possa então aplicar as ferramentas de TinyML em aplicações práticas com microcontroladores, utilizando sua placa BitDogLab.

Espero que este material de apoio enriqueça seu aprendizado e sirva como uma base sólida para você explorar mais profundamente o mundo da Aprendizagem de Máquinas e do TinyML. Recomendo que aprofunde seus estudos consultando as referências indicadas ao longo do texto, que oferecem material complementar valioso. Vamos juntos explorar o fascinante mundo do TinyML!

Referências

[1] WAARDENBURG, Pete; NICHOLAS, Daniel. TinyML Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. Sebastopol: O'Reilly Media, 2020.

[2] RASPBERRY PI LTD. RP2040 Datasheet. 2024. Disponível em: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. Acesso em: 03 set. 2024.

[3] BitDogLab. Manual BitDogLab, Disponível em: https://github.com/BitDogLab/BitDogLab/tree/main/doc. Acesso em: 03 set. 2024.

[4] EDGE IMPULSE. firmware-pi-rp2040. GitHub. Disponível em: https://github.com/edgeimpulse/firmware-pi-rp2040. Acesso em: 06 nov. 2024.

[5] Raspberry Pi. pico-tflmicro. GitHub. Disponível em: https://github.com/raspberrypi/pico-tflmicro. Acesso em: 06 nov. 2024.

[6] MORONEY, Laurence. AI and Machine Learning for Coders. Sebastopol: O'Reilly Media, 2020.

[7] DEVMEDIA. Python Tutorial. Disponível em: https://www.devmedia.com.br/python-tutorial/33274. Acesso em: 7 nov. 2024.

[8] PYTHON SOFTWARE FOUNDATION. Tutorial do Python: Documentação oficial em português. Disponível em: https://docs.python.org/pt-br/3/tutorial/. Acesso em: 7 nov. 2024.

[9] HEATON, Jeff. Applications of Deep Neural Networks with Keras. Chesterfield: Heaton Research, 2020.


embarca
tech
Residência
Tecnológica
em Sistemas
Embarcados