AULA 4
LINGUAGEM PYTHON EM APRENDIZADO DE MÁQUINA

1. Objetivos

- **Python é a linguagem de programação mais utilizada na área de aprendizado de máquina.**
- Introduzir conceito de tensores.
- Apresentar a biblioteca de funções Numpy do Python.
- Introdução à vetorização de cálculo.
- Cálculo vetorizado usando Numpy.

2. Tensores

- Tensor é um tipo de estrutura de dados que permite armazenar números em matrizes (“arrays”) de múltiplas dimensões.
- Tensores são amplamente utilizados na área de aprendizado de máquina ⇒ comum nas RNAs ter tensores de 5 ou mais dimensões.
- Importância dos tensores nas RNA é tão grande que um dos principais softwares da área é chamado TensorFlow.
- Biblioteca de funções Numpy foi desenvolvida para trabalhar com tensores.
- Existem inúmeras funções e métodos na biblioteca Numpy para operar com tensores.

Principais atributos de um tensor

- Um tensor é definido por três atributos principais:
  - Número de eixos (“rank”) ⇒ por exemplo, uma matriz possui 2 eixos, portanto, é um tensor 2D;
  - Dimensão (“shape”) ⇒ descreve quantos elementos o tensor possui em cada eixo;
  - Tipo de dados ⇒ descreve o tipo de dado que o tensor contém, por exemplo, inteiro (int8), (float32) etc. Não existem tensores de “strings” na biblioteca Numpy.

3. Tipos de tensores mais usados

Escalar

- Escalar ⇒ tensor de dimensão zero (0D).
- Tensor que armazena somente um número inteiro, real ou complexo.
- Número de eixos de um tensor ⇒ obtido pelo método `ndim`.
- Tipo de dados do tensor ⇒ obtido pelo método `dtype`.
- Função `numpy.array` ⇒ usada para criar um tensor.
- Código do Quadro 1 apresenta exemplo de um tensor escalar.

Quadro 1. Exemplo de código para criar um tensor 0D (escalar) e aplicar alguns métodos associados.

```
# Importa biblioteca numpy
import numpy as np

# Cria tensor escalar x
x = np.array(12)

# Apresenta valor de x, número de eixos e tipo de dado
print(x)
print(x.ndim)
print(x.dtype)

# Resulatdos
12
0
int32
```

Vetor

- Vetor ⇒ tensor de uma dimensão (1D), ou de um eixo.
- Método `shape` ⇒ fornece a dimensão de um tensor (número de linhas, número de colunas etc).
- Note que o número de eixos de um tensor também é chamado de “rank”, para não confundir o seu número de elementos com o seu número de eixos.
- Vetores são a forma mais comum de dados ⇒ em um conjunto de dados, cada exemplo é em geral armazenado como um vetor coluna ou linha. Se colocarmos todos os exemplos juntos (colunas) temos uma matriz de dados, onde cada coluna representa um exemplo.
- Código do Quadro 2 apresenta um exemplo de um tensor de 1D.

Quadro 2. Exemplo de código para criar um tensor 1D (vetor) e aplicar alguns métodos associados.

```
# Importa biblioteca numpy
import numpy as np

# Cria vetor x
x = np.array([12., 3., 6., -1.])

# Imprime vetor, número de eixos, dimensões dos eixos e tipo de dado
print(x)
print(x.ndim)
print(x.shape)
print(x.dtype)

# Resulatdos
[12 3 6 -1]
1
(4,)
float64
```

- O vetor do Quadro 2 tem dimensão 4x1 ⇒ ressalta-se, novamente, para não confundir a dimensão de um tensor com o número de eixos do tensor.

Matrizes

- Matriz ⇒ tensor de duas dimensões (2D), ou de dois eixos:
  - Primeiro eixo ⇒ eixo[0] representa as linhas;
  - Segundo eixo ⇒ eixo[1] representa as colunas.
- Uma matriz pode ser considerada como sendo um "array" de vetores.

- Um exemplo típico de dados de 2D é uma imagem em tons de cinza ⇒ onde o primeiro eixo representa a altura da imagem e o segundo eixo a largura. Cada elemento da matriz consiste de um número que representa a intensidade luminosa daquele local da imagem.
- Por exemplo, um conjunto de 128 imagens em tons de cinza, cada uma com dimensão 32x32 pode ser armazenado em um tensor de 3 eixos de dimensão (128, 32, 32) ou (32, 32, 128).
- Código do Quadro 3 apresenta exemplo de um tensor de 2D.

Quadro 3. Exemplo de código para criar um tensor 2D (matriz) e aplicar alguns métodos associados.

```
# Cria matriz A
A = np.array([[1,2,3],[2,3,4],[3,4,5],[4,5,6]], dtype='float32')

# Imprime vetor, número de eixos, dimensões dos eixos e tipo de dado
```

```
print(A)
print(A.ndim)
print(A.shape)
print(A.dtype)

# Resulatdos
[[1. 2. 3.]
 [2. 3. 4.]
 [3. 4. 5.]
 [4. 5. 6.]]
2
(4, 3)
float32
```

- Matriz do Quadro 3 tem 2 eixos e dimensão (4, 3), ou seja, 4 linhas e 3 colunas.

Tensores 3D ou de mais eixos

- Tensor 3D ⇒ tensor de três eixos:
  - Primeiro eixo ⇒ eixo[0] representa as linhas;
  - Segundo eixo ⇒ eixo[1] representa as colunas;
  - Terceiro eixo ⇒ eixo[2] representa a profundidade.
- Se juntarmos várias matrizes em um único tensor teremos um tensor de 3 eixos (3D) ⇒ que pode ser visualizado como um cubo ou um paralelepípedo.
- Um tensor 3D pode ser considerado como sendo um "array" de matrizes.
- Se juntarmos vários tensores 3D em um único tensor teremos um tensor 4D e, assim, por diante.
- Um exemplo típico de dados 3D é uma imagem colorida ⇒ primeiro eixo representa a altura, o segundo eixo a largura e o terceiro eixo a cor.
- Por exemplo, um conjunto de 128 imagens coloridas, cada uma com dimensão (32, 32, 3) pode ser armazenado em um tensor de 4 eixos de dimensão (32, 32, 3, 128) ou (128, 32, 32, 3).
- Um exemplo típico de dados 4D é um vídeo que consiste de um "array" de imagens, sendo que cada imagem do vídeo possui 3 eixos. O quarto eixo de um vídeo representa o tempo.
- Por exemplo, um vídeo de 60 segundos, amostrado com 10 quadros por segundo, se cada quadro do vídeo é uma imagem colorida de dimensão (256, 256, 3), esse vídeo pode ser armazenado em um tensor de 4 eixos de dimensão (256, 256, 3, 600).
- O código do Quadro 4 apresenta exemplo de um tensor de 3D.

Quadro 4. Exemplo de código para criar um tensor 3D e aplicar alguns métodos associados.

```
# Cria tensor T
T = np.array([[[1,-1],[2,-2],[3,-3]],[[2,-2],[3,-3],[4,-4]],[[3, -3],[4, -
4],[5,-5]]])

# Imprime vetor, número de eixos, dimensões dos eixos e tipo de dado
print(T)
print(T.ndim)
print(T.shape)
print(T.dtype

# Resulatdos
[[[ 1 -1]
  [ 2 -2]
  [ 3 -3]]

 [[ 2 -2]
  [ 3 -3]
  [ 4 -4]]

 [[ 3 -3]
  [ 4 -4]
  [ 5 -5]]]
3
(3, 3, 2)
int32
```

- Tensor do Quadro 4 tem 3 eixos e dimensão (3, 3, 2), ou seja, 3 linhas, 3 colunas e profundidade 2.

4. Acesso aos elementos de um tensor

- Em algumas situações pode ser necessário realizar operações com somente parte de um tensor.
- É possível selecionar somente alguns elementos de um tensor numpy.
- Seja um tensor de dimensão (100, 32, 64), que pode ser, por exemplo, 100 imagens em tons de cinza, cada uma com tamanho de 32 por 64 pixels. O código do Quadro 5 mostra alguns exemplos de como selecionar elementos desse tensor.

Quadro 5. Exemplo de código para selecionar partes de um tensor.

```
# Importa bilblioteca numpy
import numpy as np

# Impressão das dimensões do tensor com as imagens
print(imagens.shape)

# Seleciona as imagens de números 10 a 99
```

```
imagens1 = imagens[10:100,:,:]
print(imagens1.shape)

# Seleciona uma parte da imagem com os 16x16 primeiros pixels
imagens2 = imagens[:,:16,:16]
print(imagens2.shape)

# Seleciona a parte central das imagens com tamanho de 16x24 pixels
imagens3 = imagens[:,8:-8,20:-20]
print(imagens3.shape)

# Resultados
(100, 32, 64)
(90, 32, 64)
(100, 16, 16)
(100, 16, 24)
```

- Note que esse tipo de operação funciona exatamente igual à seleção de elementos em uma lista Python.

5. Vetorização de cálculo

- Todas as operações em uma RNA podem ser reduzidas para um pequeno conjunto de operações usando tensores ⇒ por exemplo, é possível adicionar, multiplicar e realizar outras operações mais complexas diretamente com tensores sem a necessidade de usar comandos de repetição para lidar com cada elemento do tensor isoladamente.

- **O que é vetorização de cálculo?**
  - Uma forma de realizar cálculos numéricos com computador evitando ao máximo comandos de repetição explícitos, como por exemplo, "for-loops".
  - Os cálculos são realizados diretamente com tensores usando funções desenvolvidas especialmente para isso.

- Em Python cálculo vetorizado de tensores é realizado pelas funções da biblioteca Numpy.
  - Biblioteca Numpy possui diversas funções matemáticas que realizam cálculos diretamente em tensores.
  - Cuidado ⇒ algumas funções da biblioteca Numpy realizam os cálculos elemento por elemento e outras realizam o cálculo considerando as regras da álgebra linear para multiplicar matrizes e vetores.
  - Outra biblioteca de funções Python que implementa cálculo vetorizado é o **TensorFlow** ⇒ uma das ferramentas mais usadas para desenvolver sistemas de inteligência artificial com redes neurais deep-learning.

- Cálculo vetorizado é muito mais rápido do que não vetorizado e é facilmente implementado usando múltiplas CPUs e em GPU (Graphics Processing Unit – Placa de Vídeo).

- Existem quatro tipos de operações básicas com tensores:
  - Operação elemento por elemento (“element-wise”);
  - Produto de tensores (“dot-product”);
  - Redimensionamento de tensores (“reshaping”);
  - Ajuste de dimensões (“Broadcasting”).

6. Operação elemento por elemento (“elemento-wise”)

- Operações elemento por elemento de tensores são operações aplicadas independentemente em cada elemento do tensor.
- Duas funções convenientes da biblioteca Numpy para criar tensores com todos elementos iguais a zero e um, são:
  - Tensor de zeros ⇒ `u = zeros((n,m))`, onde `(n, m)` é uma `tuple, sendo que n` = número de linhas e `m` = número de colunas;
  - Tensor de uns ⇒ `v = ones((n, m))`.
- O código do Quadro 6 apresenta uma implementação ingênua para calcular a soma e o produto de dois vetores elemento por elemento.

Quadro 6. Código para implementação ingênua da soma e do produto realizadas elemento por elemento de dois vetores.

```
# Importa bilblioteca numpy
import numpy as np

# Cria e inicializa tensors numpy
x = np.array([1.,2.,3.,4.,5.]) # cria vetor x 1D
y = np.array([-1,-2,-3,-4,-5]) # cria vetor y 1D
z = np.zeros(x.shape) # inicializa vetor 1D para armazenar soma x + y
w = np.zeros(x.shape) # inicializa vetor 1D para armazenar produto x*y
n = x.shape[0] # salva dimensão dos vetores

# Comando de repetição para cada elemento dos vetores
for i in range(n):
    z[i] = x[i] + y[i]
    w[i] = x[i]*y[i]

# Gera resultados
print(z)
print(w)

# Resultados
[0. 0. 0. 0. 0.]
[ -1.  -4.  -9. -16. -25.]
```

- O código do Quadro 7 apresenta os mesmos cálculos do Quadro 6, mas usando funções da biblioteca Numpy com tensores ⇒ além de ser muito mais fácil é muito mais rápido computacionalmente.

Quadro 7. Código para implementação vetorizada da soma e produto elemento por elemento de dois vetores.

```
# Operação vetorizada de soma de dois vetores elemento-por-elemento
z = x + y

# Operação vetorizada de produto dois vetores elemento-por-elemento
w = x*y

# Gera resultados
print(z)
print(w)

# Resultados
[0. 0. 0. 0. 0.]
[ -1.  -4.  -9. -16. -25.]
```

- Ressalta-se que para realizar operações elemento por elemento de tensores os dois devem ter as mesmas dimensões.

- Outras funções comuns que implementam operações elemento por elemento da biblioteca Numpy:

  Dado o vetor numpy: $\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$

  - Função exponencial ⇒ $\mathbf{u}$ = `exp(`$\mathbf{v}$`)` = $\begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

  - Função logaritmo ⇒ $\mathbf{u}$ = `log(`$\mathbf{v}$`)` = $\begin{bmatrix} \log(v_1) \\ \vdots \\ \log(v_n) \end{bmatrix}$

  - Função valor absoluto ⇒ $\mathbf{u}$ = `abs(`$\mathbf{v}$`)` = $\begin{bmatrix} |v_1| \\ \vdots \\ |v_{n1}| \end{bmatrix}$

- Potência $\Rightarrow$ `u =v**2` = $\begin{bmatrix} v_1^2 \\ \vdots \\ v_n^2 \end{bmatrix}$

- Máximo e mínimo de um tensor:

$$\begin{cases} \texttt{umax = max(u)} \\ \texttt{umin = min(u)} \end{cases}$$

7. Produto de dois tensores ("dot-product")

- O produto de dois tensores é a operação mais comum e talvez a mais útil nos cálculos realizados nas RNAs.

- Essa operação é chamada de "dot-product" a partir de uma analogia com o produto escalar de dois vetores.

- A operação "dot-product" é igual ao produto escalar de dois vetores para o caso dos tensores serem 1D, ou seja, vetores.

- A operação "dot-product" é igual ao produto de uma matriz por um vetor quando um dos tensores é 2D e o outro é 1D com as dimensões corretas para poder realizar a operação.

- O código do Quadro 8 apresenta uma implementação ingênua para calcular o produto escalar de dois vetores com mesmas dimensões e o produto de uma matriz por um vetor.

Quadro 8. Implementação ingênua do produto escalar de dois vetores e do produto de uma matriz por um vetor.

```
# Importa bilblioteca numpy
import numpy as np

# Cria tensores
x = np.ones((1,3))
y = np.array([1., 2., 3.])
A = np.array([[1,1,1],[2,2,2],[3,3,3]], dtype='float32')
w = np.zeros(A.shape[0])

# Recupera dimensões dos tensores
n = x.shape[0]
m, n = A.shape

% Cálculo do produto escalar dos vetores x e y
z = 0
for i in range(n):
    z += x[i]*y[i]

% Cálculo do produto da matriz A pelo vetor y
```

```
for i in range(m):
    for j in range(n):
        w[i] += A[i][j]*y[j]

# Impressão dos resultados
print("x = ",x)
print("y = ",y)
print("A = ",A)
print("z = ",z)
print("w = ",w)

# Resultados
x = [[1. 1. 1.]]
y = [1. 2. 3.]
A = [[1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]]
z = [6.]
w = [ 6. 12. 18.]
```

- O código do Quadro 9 apresenta os cálculos do código do anterior (Quadro 8) usando a função `dot` da biblioteca Numpy.

Quadro 9. Implementação vetorizada do produto escalar de dois vetores e do produto de uma matriz por um vetor.

```
# Cálculo vetorizado do produto escalar dos vetores x e y
z = np.dot(x,y)

# Cálculo vetorizado do produto da matriz A pelo vetor y
w = np.dot(a,y)

# Impressão dos resultados
print("x = ",x)
print("y = ",y)
print("A = ",A)
print("z = ",z)
print("w = ",w)

# Resultados
x = [[1. 1. 1.]]
y = [1. 2. 3.]
A = [[1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]]
z = [6.]
w = [ 6. 12. 18.]
```

- As dimensões dos tensores devem ser compatíveis para poder realizar o produto dos mesmos. A regra utilizada segue os princípios da álgebra linear.

- Por exemplo, somente é possível realizar o produto vetorial de duas matrizes **A** e **B**, se a dimensão do segundo eixo de **A** for igual à dimensão do primeiro eixo de **B**, ou seja:

`dot(A, B)` somente é possível se `A.shape[1] = B.shape[0]` e o resultado é uma matriz de dimensões `A.shape[0]` por `B.shape[1]`.

- Pode-se calcular o produto de tensores de muitos eixos usando a função `dot`. A regra geral para a compatibilidade de dimensões é a seguinte:

  (a, b) x $(b,) \rightarrow (a,)$

  (a, b, c) x $(c,) \rightarrow (a, b)$

  (a, b, c, d) x $(d,) \rightarrow (a, b, c)$

  (a, b, c, d) x $(d, e) \rightarrow (a, b, c, e)$

  onde a, b, c, d e e são as dimensões dos eixos dos tensores.

- Note que o produto de dois tensores com número de eixos maior do que um não tem propriedade comutativa, ou seja, segue os mesmos conceitos da álgebra linear para produto de duas matrizes, assim:

  `dot(A, B) ≠ dot(B, A)`

- **Números aleatórios** são muito utilizados em redes neurais ⇒ a biblioteca Numpy possui uma função para gerar números aleatórios:

  - `random.random` ⇒ função usada para criar um tensor de números aleatórios com distribuição uniforme entre 0 e 1, com a dimensão desejada;
  - O código do Quadro 10 apresenta como se usa essa função.

- O código do Quadro 10 apresenta um exemplo do cálculo vetorizado da propagação para frente da camada l de uma RNA.

  Equações da propagação para frente na l-ésima camada:

$$\begin{cases} \mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{[l]} = \sigma(\mathbf{z}^{[l]}) \end{cases}$$

  onde:

  dimensão de $\mathbf{W}^{[l]}$ = $(n^{[l]}, n^{[l-1]})$;

  dimensão de $\mathbf{a}^{[l-1]} = (n^{[l-1]}, 1)$;

  dimensão de $\mathbf{b}^{[l]}$ = dimensão de $\mathbf{a}^{[l]}$ = dimensão de $\mathbf{z}^{[l]} = (n^{[l]}, 1)$.

Quadro 10. Implementação do cálculo vetorizado da propagação para frente em uma camada de uma RNA.

```
# Importa bilblioteca numpy
import numpy as np

# Função para cálculo da propagação para frente da camada l
def frente(W,b,a_ant):
    z = np.dot(W,a_ant) + b
    a = 1/(1 + np.exp(-z))
    return a

# Inicialização dos parâmetros da camada da RNA
W = np.random.random((4,3))
b = np.random.random((4,1))
a_ant = np.ones((3,1))

# Calculo da propagação para frente na camada
a = frente(W,b,a_ant)

# Apresenta resultados
print("W = ",W)
print("b = ",b)
print("a = ",a)

# Resultados
W = [[0.91402442 0.38210853 0.46200102]
 [0.02249424 0.24887291 0.53957662]
 [0.1918867  0.86980668 0.08368954]
 [0.16724615 0.26109255 0.45316348]]
b = [[0.32582427]
 [0.37940423]
 [0.0047974 ]
 [0.67165663]]
a = [[0.8893342]
 [0.7668033]
 [0.75954385]
 [0.82536949]]
```

8. Redimensionamento de tensores (“reshaping”)

- Uma operação essencial de tensores muito usada nos cálculos envolvidos em uma RNA é o seu redimensionamento.
- Redimensionar um tensor significa rearranjar suas linhas e colunas para se obter um novo tensor com novos eixos de diferentes dimensões.
- Obviamente que o tensor redimensionado tem os mesmos elementos que o tensor original.
- Um caso especial de redimensionamento de tensores é a sua transposição ⇒ transpor uma matriz significa trocar suas linhas por suas colunas.

- O código do Quadro 11 apresenta alguns exemplos de redimensionamento de tensores usando as funções `reshape` e `transpose` da bilbioteca Numpy.

Quadro 11. Exemplos de redimensionamento e transposição de tensores.

```
# Cria tensor x 2D
x = np.array([[0., 1.],[2., 3.],[4., 5.]])
print("x = ",x)
print("dimensão de x =", x.shape)

# Redimensiona tensor x (seguindo as suas linhas)
y = x.reshape((6,1))
print("y = ", y)
print("dimensão de y =", y.shape)

# Redimensiona tensor x (seguindo as suas colunas)
z = x.reshape((6,1), order='F')
print("z = ", z)
print("dimensão de z =", z.shape)

# Cria tensor w pelo redimensionamento do tensor x
w = x.reshape((2,3))
print("w = ", w)
print("dimensão de w =", w.shape)

# Transpõe tensor w
w = np.transpose(w)
print("w transposto = ", w)
print("dimensão de w transposto =", w.shape)

# Resultados
x =  [[0. 1.]
 [2. 3.]
 [4. 5.]]
dimensão de x = (3, 2)
y =  [[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]]
dimensão de y = (6, 1)
z =  [[0.]
 [2.]
 [4.]
 [1.]
 [3.]
 [5.]]
dimensão de z = (6, 1)
w =  [[0. 1. 2.]
 [3. 4. 5.]]
dimensão de w = (2, 3)
w transposto =  [[0. 3.]
 [1. 4.]
 [2. 5.]]
dimensão de w transposto = (3, 2)
```

- Observa-se que o padrão da função `reshape` é seguir as linhas do tensor, mas isso pode ser alterado para seguir as colunas usando a opção `order` (ver Quadro 11).

9. Ajuste automático de dimensões de tensores ("broadcasting")

- Ao somar e multiplicar tensores elemento por elemento o número de eixos e as dimensões de cada eixo devem ser iguais nos dois tensores.

- Quando não existe nenhuma ambigüidade em uma operação elemento por elemento de dois tensores as funções da biblioteca Numpy ajustam as dimensões dos tensores automaticamente.

- Exemplos:

1) $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$

2) $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} + 100 = \begin{bmatrix} 101 & 102 & 103 \end{bmatrix}$

3) $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$

4) $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 \\ 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$

- **Regra geral do "broadcasting":**

  - Vetor (m, 1) +/–/*/÷ escalar = vetor (m, 1) – exemplo 1;
  - Vetor (1, n) +/–/*/÷ escalar = vetor (1, n) – exemplo 2;
  - Matriz (m, n) +/–/*/÷ vetor (1, n) = matriz (m, n) – exemplo 3;
  - Matriz (m, n) +/–/*/÷ vetor (m, 1) = matriz (m, n) – exemplo 4.