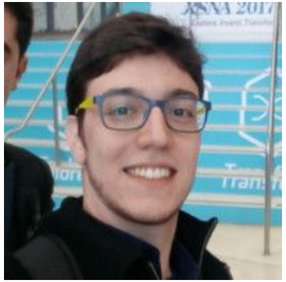
Redes Neurais Profundas para aplicações de Visão Computacional (RNP-VC)

Lucas Pereira, Rafael Teixeira, Lucas Assis, Anderson Soares
Instituto de Informática
Universidade Federal de Goiás (UFG)

Instrutores



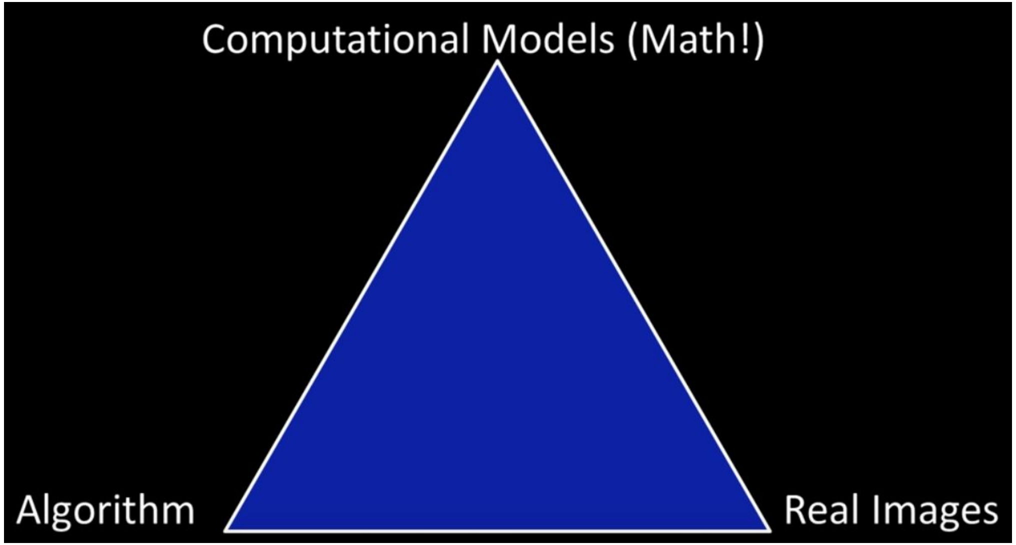
Sumário

- O que o curso é e o que não é
- Cronograma e avaliação
- Revisão de tensores

O que o curso não é:

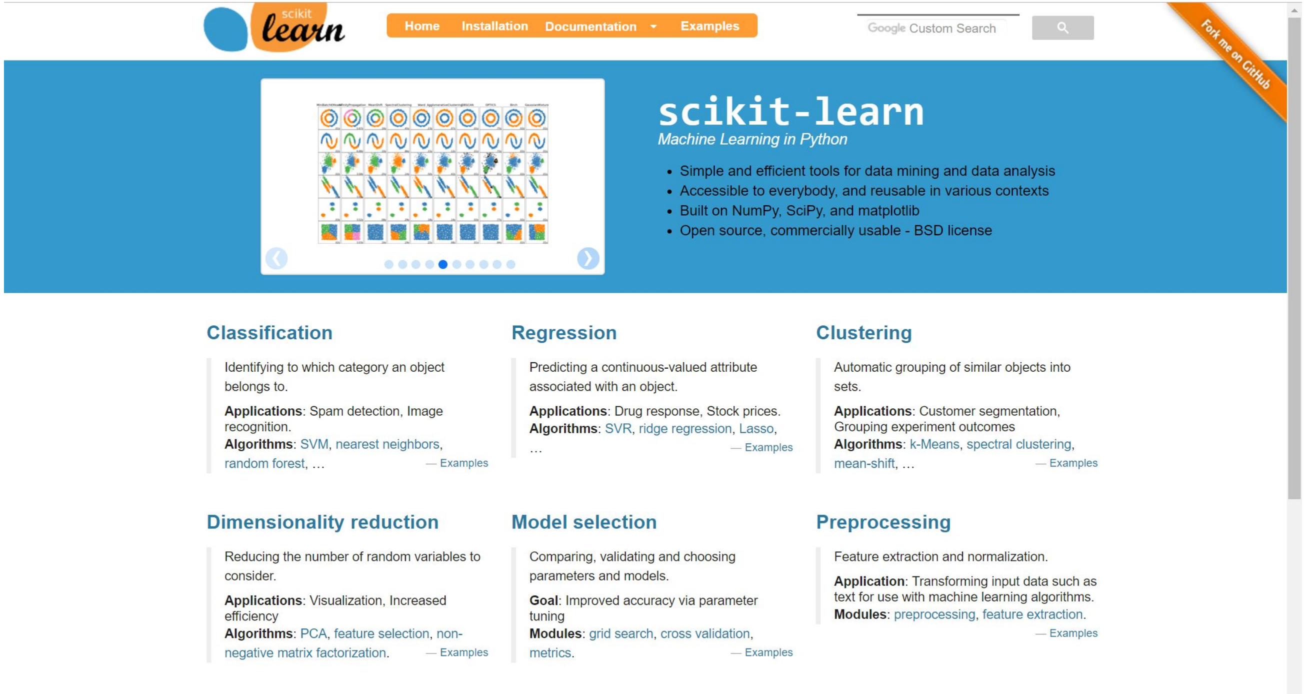
- Um curso de visão computacional

IMAGE PROCESSING
CAMERA MODELS AND VIEWS
FEATURES AND MATCHING
LIGHTNESS AND BRIGHTNESS
IMAGE MOTION
MOTION AND TRACKING
CLASSIFICATION AND RECOGNITION
MISCELLANEOUS OPERATIONS
HUMAN VISION



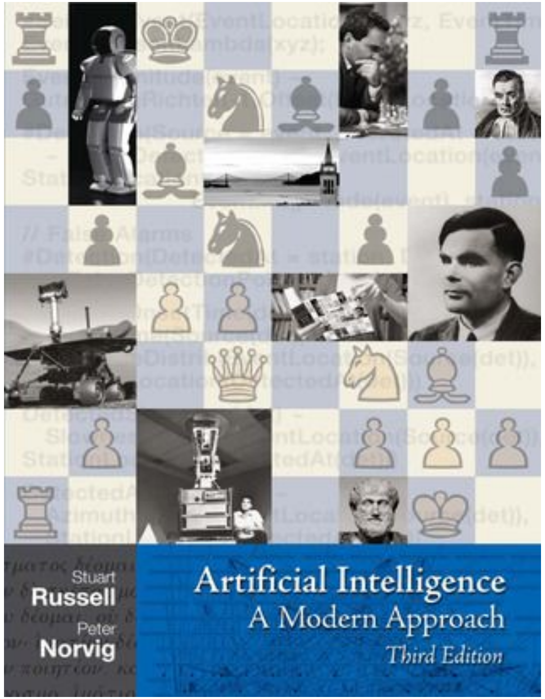
O que o curso não é:

- Um curso de Machine Learning



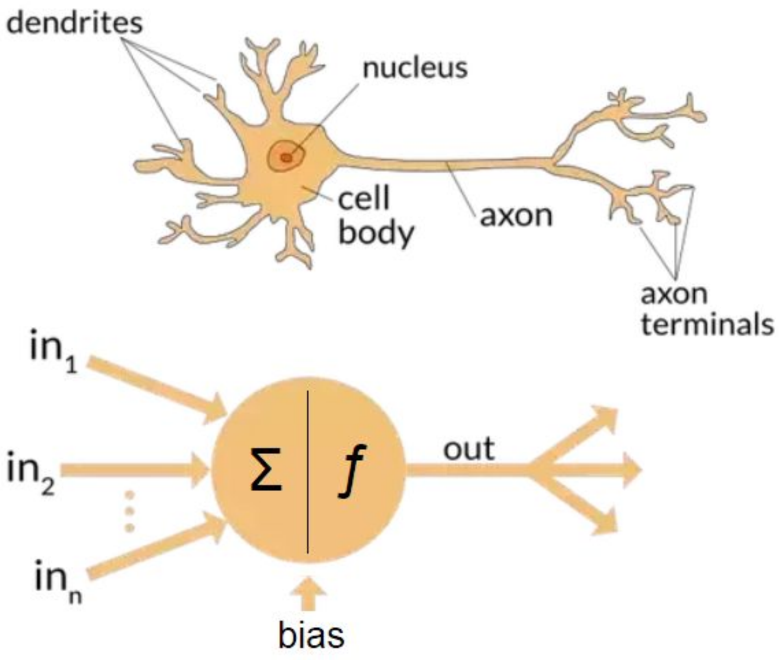
O que o curso não é:

- Um curso de Inteligência Artificial



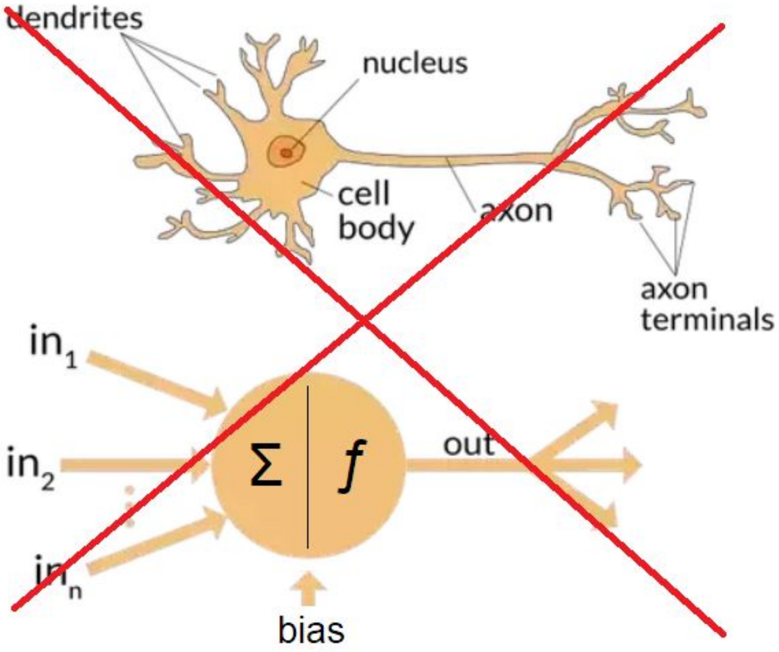
O que o curso não é:

- Um curso de neurociência



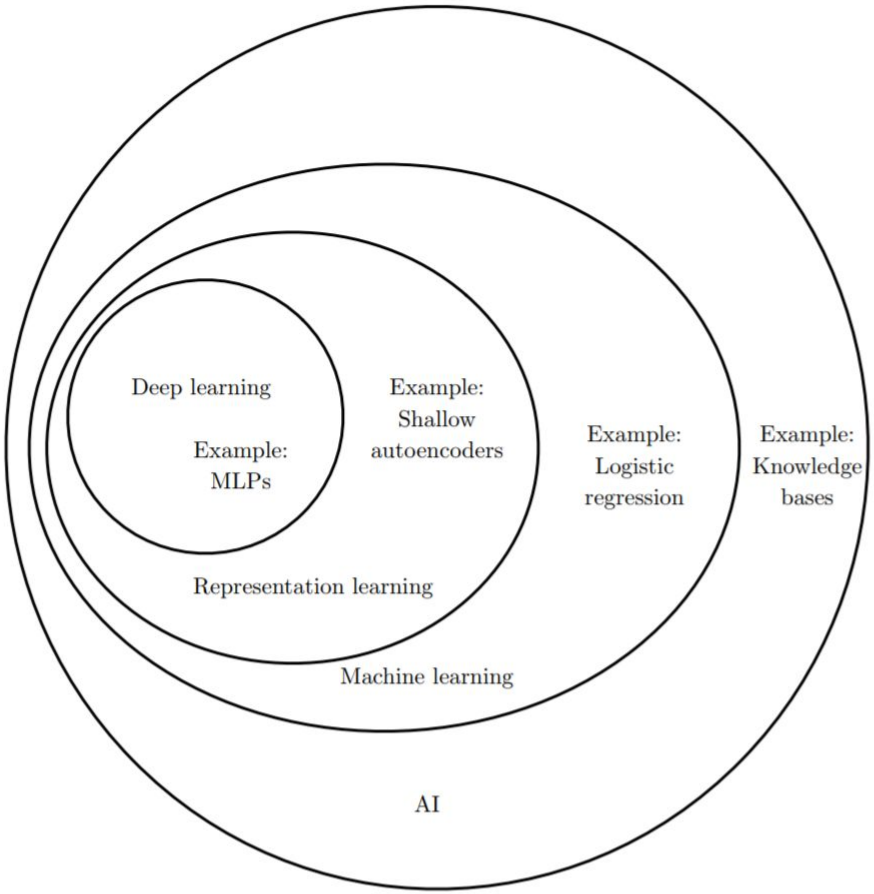
O que o curso não é:

- Um curso de neurociência



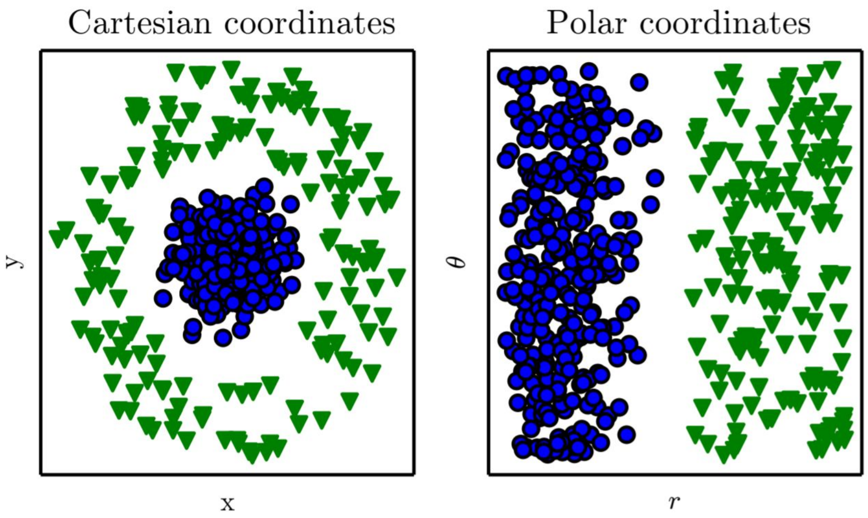
O que o curso é:

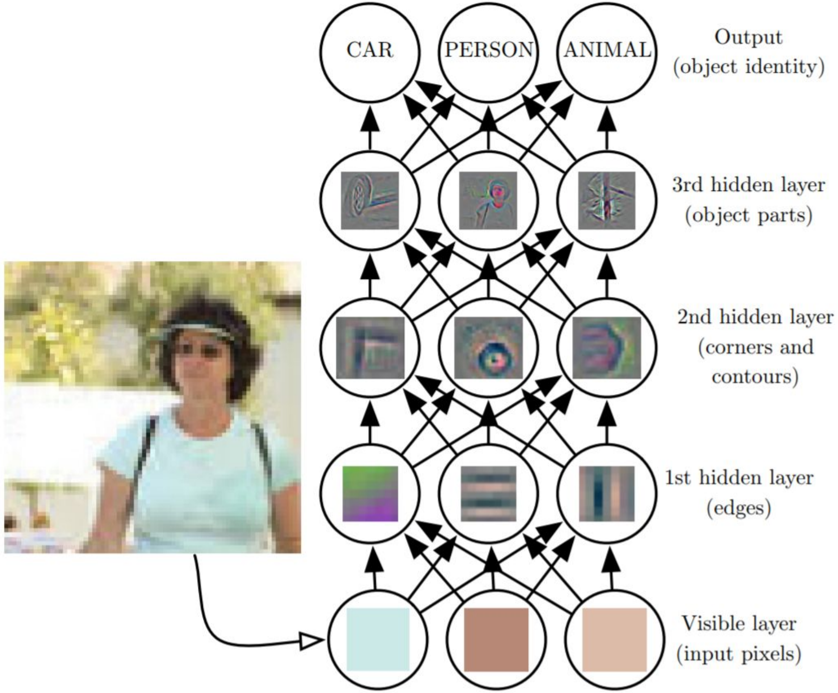
- Introdução a Redes Neurais Profundas (RNP)
- Aplicação de RNP em tarefas de Visão Computacional



O que o curso é:

- A importância do aprendizado de representações
- Introdução ao conceito de composição de representações





Cronograma

- Aula 0-0: Visão geral sobre o curso
- Aula 0-1: Revisão de tensores
- Aula 1: Introdução a ML e VC
- Aula 2: Introdução a Redes Neurais
- Aula 3: Desenvolvimento de modelos - parte 1
- Aula prática
- Aula 4: Desenvolvimento de modelos - parte 2
- Aula 5: Transfer learning e zoológico de modelos
- Aula prática
- Aula 6: Detecção de objetos
- Aula 7: Segmentação semântica
- Aula 8: Aplicações com dados sequenciais
- Aula prática
- Aula 9: Modelos generativos
- Aula 10: Visualização e interpretabilidade

Atividades avaliativas

- Participação das aulas práticas
- Prova teórica (não espere questões fáceis)
  - Provas de turmas anteriores duraram 24h… 26h…
- Projeto final para apresentação em workshop
  - Resolva o problema de alguém disposto a pagar pelo que você fez
  - Traremos pessoas de mercado para avaliar a solução

Atividades avaliativas

- Entre para a história (se puder…)

Ranking histórico da disciplina de Deep Learning			
Posição	**Nome**	**Turma**	**Nota**
1	José Adenaldo	2017/2	9,5
2	Manoel Veríssimo	2018/1	9,2
3	Gabriel Horikawa	2018/1	9
4	Rafael Teixeira	2017/1	8,8
5	Leandro Leal	2017/2	8,7
6	Pedro Vitor	2017/1	8,7
7	Larissa Moraes	2017/1	8,6
8	Alexandre Barbosa	2016/2	8,6
9	Vinícius Araújo	2017/1	8,5
	Thiago Mendes	2018/1	8,5
	Fred Oliveira	2018/1	8,5

Aula 0-1: Revisão de tensores: Algebra Linear

- Escalar x Vetor x Matriz

$$x \qquad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} X_{0,0} & X_{0,1} & \cdots & X_{0,n-1} \\ X_{1,0} & X_{1,1} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ X_{m-1,0} & \cdots & \cdots & X_{m-1,n-1} \end{bmatrix}$$

$$x \in \mathbb{R} \qquad \mathbf{x} \in \mathbb{R}^n \qquad \mathbf{X} \in \mathbb{R}^{m,n}$$

Algebra Linear

- Tensor?

Algebra Linear

- Tensor $\mathbf{X} \in \mathbb{R}^{n_0, n_1, \cdots, n_k}$

Quiz:

- Tensor de 0 dimensões?
- Tensor de 1 dimensão?
- Tensor de 2 dimensões?

Algebra Linear

- Tensor $\mathbf{X} \in \mathbb{R}^{n_0, n_1, \cdots, n_k}$

Quiz:

- Tensor de 0 dimensões? *Escalar*
- Tensor de 1 dimensão? *Vetor*
- Tensor de 2 dimensões? *Matriz*

Algebra Linear

- Operações com matrizes
  - Soma e subtração

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \Leftrightarrow C_{i,j} = A_{i,j} + B_{i,j}$$

  - Transposta

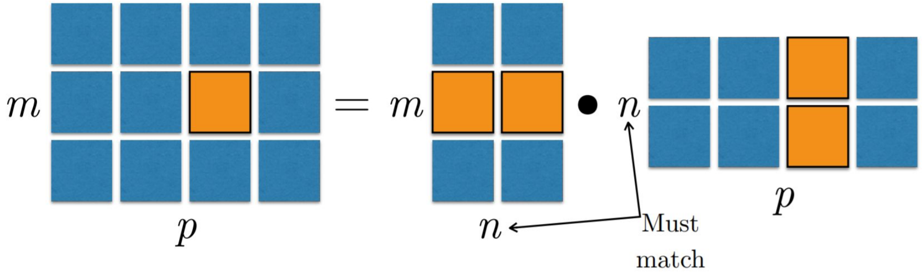
$$(\boldsymbol{A}^\top)_{i,j} = A_{j,i}$$

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

  - Produto (matmul ou dot product)

$$\mathbf{C} = \mathbf{AB} \Leftrightarrow C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

$$\mathbf{AB} \neq \mathbf{BA}$$



Algebra Linear

- Divisão?

Algebra Linear

- Divisão = produto do inverso!

$$\mathbf{A}/\mathbf{B} = \mathbf{A}\mathbf{B}^{-1}$$

- Inverso*

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n \qquad \mathbf{I}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 \end{bmatrix}, \mathbf{I}_n \in \mathbb{R}^{n,n}$$

* nem toda matriz tem inverso!

Algebra Linear

- Operações com tensores
  - Operações elemento por elemento (element-wise)

$$\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n_0, n_1, \cdots, n_k}:$$
$$C_{i_0, i_1, \cdots, i_k} = A_{i_0, i_1, \cdots, i_k} + B_{i_0, i_1, \cdots, i_k}$$
$$C_{i_0, i_1, \cdots, i_k} = A_{i_0, i_1, \cdots, i_k} B_{i_0, i_1, \cdots, i_k}$$

  - Operações com escalares (broadcast)

$$\mathbf{A}, \mathbf{C} \in \mathbb{R}^{n_0, n_1, \cdots, n_k}, b \in \mathbb{R}:$$
$$C_{i_0, i_1, \cdots, i_k} = A_{i_0, i_1, \cdots, i_k} + b$$
$$C_{i_0, i_1, \cdots, i_k} = b A_{i_0, i_1, \cdots, i_k}$$

Algebra Linear

- Operações com tensores
  - Transposta

numpy.transpose(*a*, *axes=None*) [source]

Permute the dimensions of an array.

Parameters:	**a** : *array_like* Input array. **axes** : *list of ints, optional* By default, reverse the dimensions, otherwise permute the axes according to the values given.
Returns:	**p** : *ndarray* *a* with its axes permuted. A view is returned whenever possible.

```
>>> x = np.ones((1, 2, 3))
>>> np.transpose(x, (1, 0, 2)).shape
(2, 1, 3)
```

Algebra Linear

- Operações com tensores
  - Transposta

`torch.transpose(`*`input, dim0, dim1`*`)` → Tensor

Returns a tensor that is a transposed version of `input`. The given dimensions `dim0` and `dim1` are swapped.

The resulting `out` tensor shares it's underlying storage with the `input` tensor, so changing the content of one would change the content of the other.

Parameters

- **input** (*Tensor*) – the input tensor
- **dim0** (*int*) – the first dimension to be transposed
- **dim1** (*int*) – the second dimension to be transposed

Algebra Linear

- Operações com tensores
    - Transposta

```
tf.transpose(
    a,
    perm=None,
    name='transpose',
    conjugate=False
)
```

Permutes the dimensions according to `perm`.

The returned tensor's dimension i will correspond to the input dimension `perm[i]`. If `perm` is not given, it is set to (n-1...0), where n is the rank of the input tensor. Hence by default, this operation performs a regular matrix transpose on 2-D input Tensors. If conjugate is True and `a.dtype` is either `complex64` or `complex128` then the values of `a` are conjugated and transposed.

Algebra Linear

- Multiplicação de tensores (matmul ≠ dot product)

```
>>> a = np.ones([9, 5, 7, 4])
>>> c = np.ones([9, 5, 4, 3])
>>> np.dot(a, c).shape
(9, 5, 7, 9, 5, 3)
>>> np.matmul(a, c).shape
(9, 5, 7, 3)
>>> # n is 7, k is 4, m is 3
```

Algebra Linear

- Multiplicação de tensores (matmul ≠ dot product)
  - `tf.matmul`

```
tf.linalg.matmul(
    a,
    b,
    transpose_a=False,
    transpose_b=False,
    adjoint_a=False,
    adjoint_b=False,
    a_is_sparse=False,
    b_is_sparse=False,
    name=None
)
```

The inputs must, following any transpositions, be tensors of rank >= 2 where the inner 2 dimensions specify valid matrix multiplication arguments, and any further outer dimensions match.

Algebra Linear

- Multiplicação de tensores (matmul ≠ dot product)

```
>>> # vector x vector
>>> tensor1 = torch.randn(3)
>>> tensor2 = torch.randn(3)
>>> torch.matmul(tensor1, tensor2).size()
torch.Size([])
>>> # matrix x vector
>>> tensor1 = torch.randn(3, 4)
>>> tensor2 = torch.randn(4)
>>> torch.matmul(tensor1, tensor2).size()
torch.Size([3])
>>> # batched matrix x broadcasted vector
>>> tensor1 = torch.randn(10, 3, 4)
>>> tensor2 = torch.randn(4)
>>> torch.matmul(tensor1, tensor2).size()
torch.Size([10, 3])
>>> # batched matrix x batched matrix
>>> tensor1 = torch.randn(10, 3, 4)
>>> tensor2 = torch.randn(10, 4, 5)
>>> torch.matmul(tensor1, tensor2).size()
torch.Size([10, 3, 5])
>>> # batched matrix x broadcasted matrix
>>> tensor1 = torch.randn(10, 3, 4)
>>> tensor2 = torch.randn(4, 5)
>>> torch.matmul(tensor1, tensor2).size()
torch.Size([10, 3, 5])
```

Algebra Linear

- Broadcasting

1. they are equal, or
2. one of them is 1

```
>>> x=torch.empty(5,7,3)
>>> y=torch.empty(5,7,3)
# same shapes are always broadcastable (i.e. the above rules always hold)

>>> x=torch.empty((0,))
>>> y=torch.empty(2,2)
# x and y are not broadcastable, because x does not have at least 1 dimension

# can line up trailing dimensions
>>> x=torch.empty(5,3,4,1)
>>> y=torch.empty(  3,1,1)
# x and y are broadcastable.
# 1st trailing dimension: both have size 1
# 2nd trailing dimension: y has size 1
# 3rd trailing dimension: x size == y size
# 4th trailing dimension: y dimension doesn't exist

# but:
>>> x=torch.empty(5,2,4,1)
>>> y=torch.empty(  3,1,1)
# x and y are not broadcastable, because in the 3rd trailing dimension 2 != 3
```

Algebra Linear

- Broadcasting

1. they are equal, or
2. one of them is 1

```
>>> x = np.arange(4)
>>> xx = x.reshape(4,1)
>>> y = np.ones(5)
>>> z = np.ones((3,4))
>>> y.shape
(5,)
>>> x.shape
(4,)
>>> x + y
ValueError: operands could not be broadcast together with shapes (4,) (5,)
>>> xx.shape
(4, 1)
>>> xx + y
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.,  4.]])
```

Algebra Linear

- Por que matrizes?

Algebra Linear

- Por que matrizes?
  - Sistemas lineares

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases} \Leftrightarrow \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{A} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \mathbf{b} = \begin{bmatrix} c \\ f \end{bmatrix}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

$$\boldsymbol{I}_n\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

Algebra Linear

- Por que matrizes?
  - Sistemas lineares
  - Interpretação geométrica: conjunto de vetores (tamanho, direção e sentido)

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

$$\boldsymbol{A} = \boldsymbol{V}\operatorname{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$$

autovetores e autovalores
ou
decomposição de valor singular
(SVD)