UNIVERSIDAD EAFIT

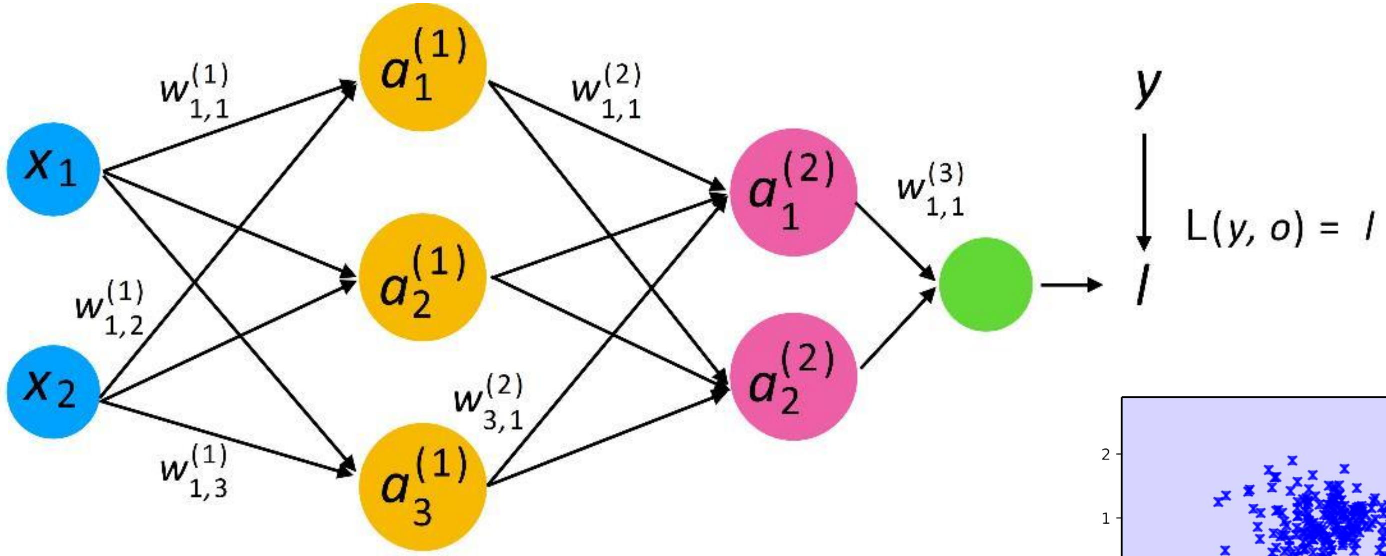
Escuela de Ciencias Aplicadas e Ingeniería

Lecture02

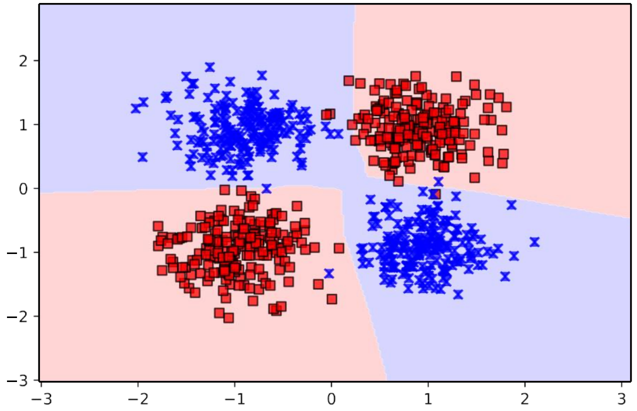
Álgebra lineal para aprendizaje profundo

Hoy: Habilidades matemáticas fundamentales para el aprendizaje profundo

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Para que podamos resolver el problema XOR, entre otras cosas…



MLP de una capa oculta con función de activación no lineal (ReLU)

Resumen de la clase

1. Tensores en el aprendizaje profundo
2. Tensores y PyTorch
3. Vectores, matrices y broadcasting
4. Convenciones de notación para redes neuronales
5. Una capa totalmente conectada (lineal) en PyTorch

Uso de tensores en el aprendizaje profundo

1. **Tensores en el aprendizaje profundo**
2. Tensores y PyTorch
3. Vectores, matrices y broadcasting
4. Convenciones de notación para redes neuronales
5. Una capa totalmente conectada (lineal) en PyTorch

Vectores, matrices y tensores – Convenciones de notación

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Scalar

(rank-0 tensor)

$x \in \mathbb{R}$

e.g.,

$x = 1.23$

Vector

(rank-1 tensor)

$\mathbf{x} \in \mathbb{R}^n$

but in this lecture, we will assume

$\mathbf{x} \in \mathbb{R}^{n \times 1}$

e.g.,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Matrix

(rank-2 tensor)

$\mathbf{X} \in \mathbb{R}^{m \times n}$

e.g.,

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{bmatrix}$$

$$\mathbf{x}^\top = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \text{ where } \mathbf{x}^\top \in \mathbb{R}^{1 \times n}$$

Vectores, matrices y tensores – Convenciones de notación

A menudo usaremos X como una convención especial para referirnos a la "matriz de diseño". Es decir, la matriz que contiene los ejemplos de entrenamiento y características (entradas)

y asumimos la estructura $\mathbf{X} \in \mathbb{R}^{n \times m}$

porque *n* se usa a menudo para referirse al número de ejemplos en la literatura de muchas disciplinas.

$$\mathbf{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \dots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \dots & x_m^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \dots & x_m^{[n]} \end{bmatrix}$$

Ej,

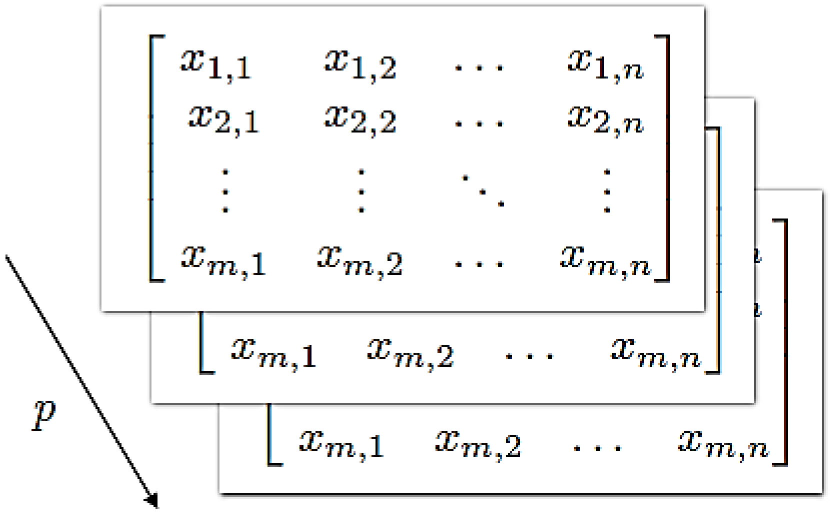
$x_2^{[1]}$ = Segundo valor de característica del primer ejemplo de entrenamiento.

Vectores, matrices y tensores – Convenciones de notación

3D Tensor

(rank-3 tensor)

$X \in \mathbb{R}^{m \times n \times p}$ (n y m son índices genéricos aquí)



...

Un ejemplo de un tensor 3D en el aprendizaje profundo

Imagen en un solo canal (color)

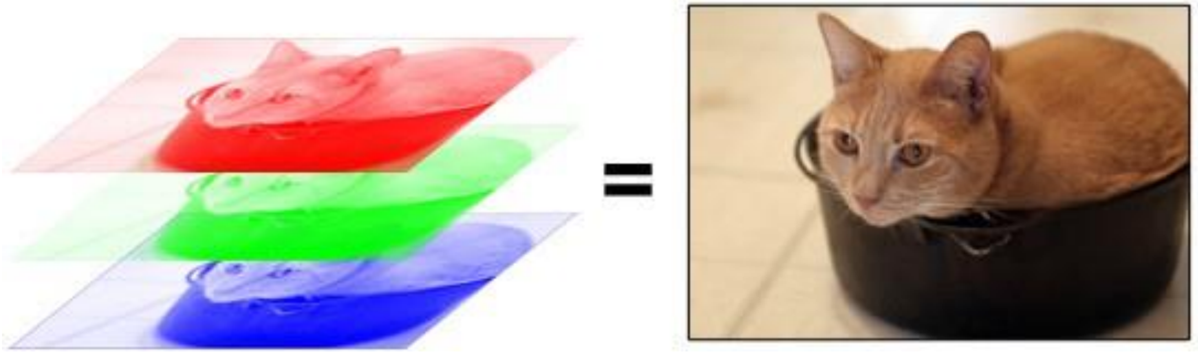
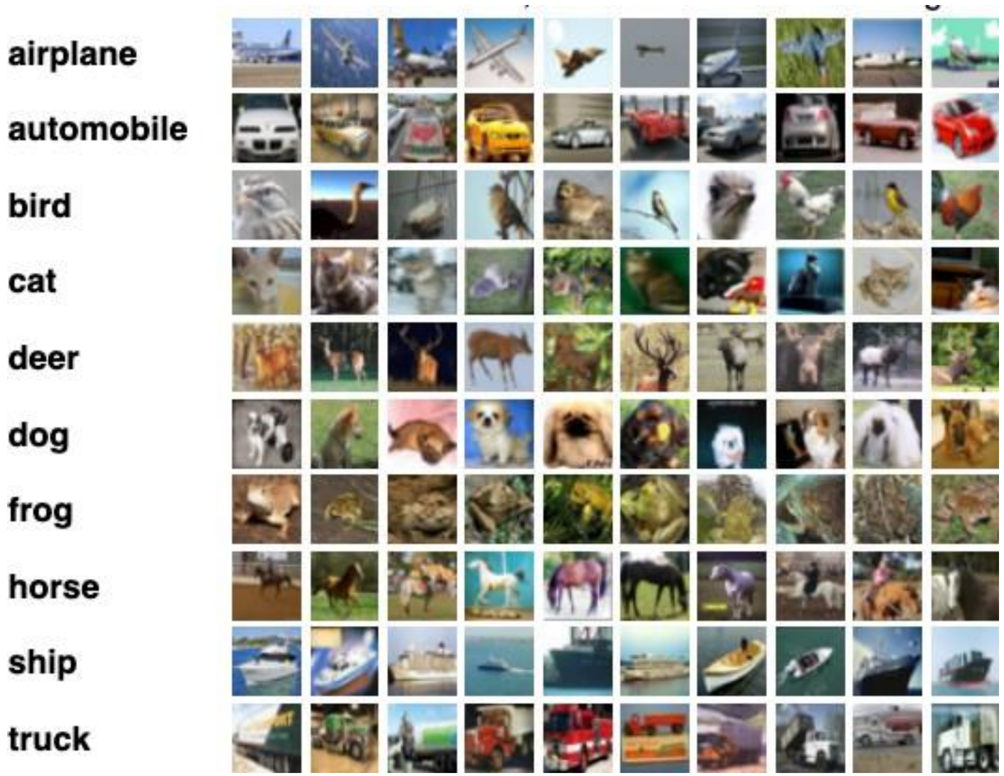


Image Source: https://code.tutsplus.com/tutorials/create-a-retro-crt-distortion-effect-using-rgb-shifting--active-3359

(tensor 3D para fines de almacenamiento como "arreglo multidimensional" y computación paralela, aún usamos matemáticas regulares de vectores y matrices)

Un ejemplo de un tensor 4D en el aprendizaje profundo

Lote (batch) de imágenes(como entrada para la red neuronal, más adelante)



https://www.cs.toronto.edu/~kriz/cifar.html

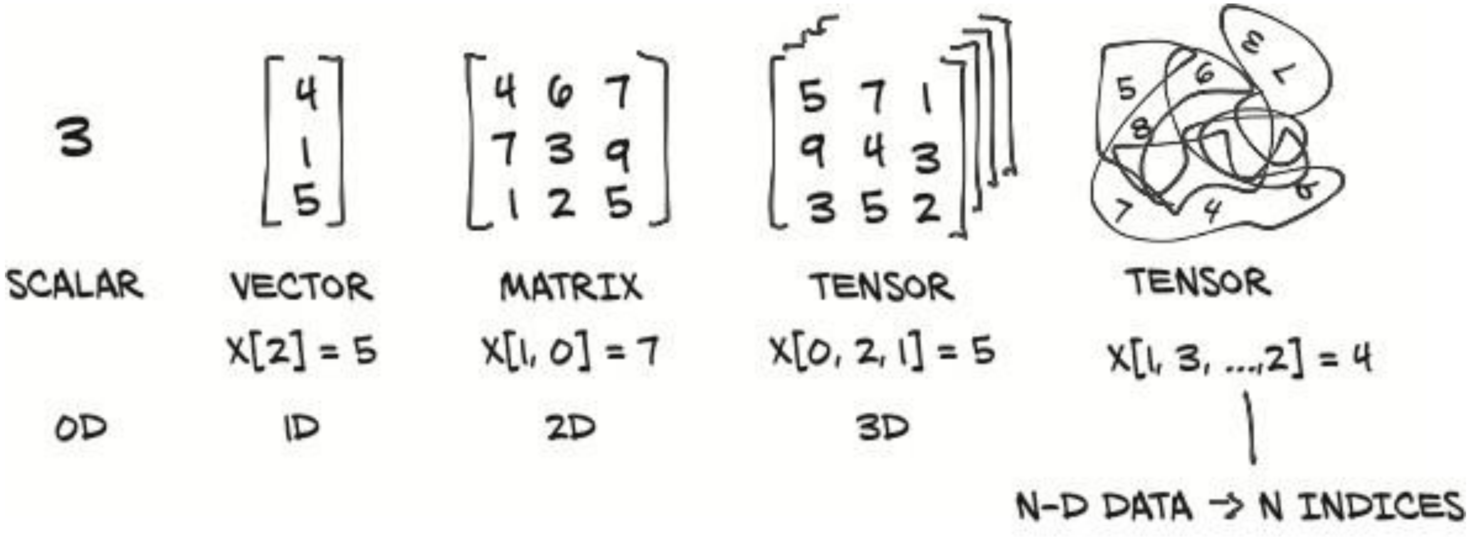
(tensor 4D para fines de almacenamiento como "arreglo multidimensional" y computación paralela, aún usamos matemáticas regulares de vectores matrices)

En el contexto de TensorFlow, NumPy, PyTorch, etc.,tensores = arreglos multidimensionales

La dimensionalidad coincide con el número de índices de .shape

```
In [1]: import torch

In [2]: t = torch.tensor([[1, 2, 3], [4, 5, 6]])

In [3]: t
Out[3]:
tensor([[1, 2, 3],
        [4, 5, 6]])

In [4]: t.shape
Out[4]: torch.Size([2, 3])

In [5]: t.ndim
Out[5]: 2

In [6]:
```

Escuela de Ciencias Aplicadas e Ingeniería



gure 3.2 Tensors are the building blocks for representing data in PyTorch.

Image source: Stevens et al.'s "Deep Learning with PyTorch"

Trabajando con tensores en PyTorch

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

1. Tensores en el aprendizaje profundo
2. **Tensores y PyTorch**
3. Vectores, matrices y broadcasting
4. Convenciones de notación para redes neuronales
5. Una capa totalmente conectada (lineal) en PyTorch

Arreglos multidimensionales como tensores

Universidad EAFIT — Escuela de Ciencias Aplicadas e Ingeniería

numpy.array / numpy.ndarray = (representación estructural de datos de un tensor)

pytorch.tensor / pytorch.Tensor = (representación estructural de datos de un tensor)

Ejemplo:

```
In [1]: import numpy as np

In [2]: a = np.array([1., 2., 3.])

In [3]: print(a.dtype)
float64

In [4]: print(a.shape)
(3,)
```

```
In [5]: import torch

In [6]: b = torch.tensor([1., 2., 3.])

In [7]: print(b.dtype)
torch.float32

In [8]: print(b.shape)
torch.Size([3])
```

La sintaxis de NumPy y PyTorch es muy similar

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

```
In [9]: a = np.array([1., 2., 3.])

In [10]: print(a.dot(a))
14.0
```

```
In [12]: print(b.matmul(b))
tensor(14.)

In [13]: b
Out[13]: tensor([1., 2., 3.])

In [14]: b.numpy()
Out[14]: array([1., 2., 3.], dtype=float32)
```

Podemos convertir, pero presta atención a los tipos por defecto

Nota: Tradicionalmente, PyTorch usaba "matmul",pero hoy en día "dot" también funciona

```
In [12]: print(b.matmul(b))
tensor(14.)
```

```
In [15]: print(b.dot(b))
tensor(14.)

In [16]: print(b @ b)
tensor(14.)
```

Tipos de datos para memorizar

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

NumPy data	Tensor data type	
numpy.uint8	torch.ByteTensor	
numpy.int16	torch.ShortTensor	
numpy.int32	torch.IntTensor	
numpy.int	torch.LongTensor	
numpy.int64	torch.LongTensor	default int in NumPy & PyTorch
numpy.float16	torch.HalfTensor	
numpy.float32	torch.FloatTensor	default float in PyTorch
numpy.float	torch.DoubleTensor	
numpy.float64	torch.DoubleTensor	default float in NumPy

- Ej., int32 significa entero de 32 bits
- Los flotantes de 32 bits son menos precisos que los de 64 bits, pero para redes neuronales, no importa mucho
- Para GPUs regulares, usualmente queremos flotantes de 32 bits (vs 64 bits) para mayor rapidez

Especifica el tipo al momento de la construcción

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

```
In [21]: c = torch.tensor([1., 2., 3.], dtype=torch.float)

In [22]: c.dtype
Out[22]: torch.float32

In [23]: c = torch.tensor([1., 2., 3.], dtype=torch.double)

In [24]: c.dtype
Out[24]: torch.float64

In [25]: c = torch.tensor([1., 2., 3.], dtype=torch.float64)

In [26]: c.dtype
Out[26]: torch.float64
```

También puedes cambiar los tipos después o sobre la marcha si es necesario

UNIVERSIDAD EAFIT

Escuela de Ciencias Aplicadas e Ingeniería

```
In [27]: d = torch.tensor([1, 2, 3])

In [28]: d.dtype
Out[28]: torch.int64

In [29]: e = d.double()

In [30]: e.dtype
Out[30]: torch.float64

In [31]: f = d.float64()
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-b3b070130d25> in <module>
----> 1 f = d.float64()

AttributeError: 'Tensor' object has no attribute 'float64'

In [32]: f = d.to(torch.float64)

In [33]: f.dtype
Out[33]: torch.float64
```

Entonces, ¿por qué no simplemente usar NumPy?

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- PyTorch tiene soporte para GPU:
    A. Podemos cargar el conjunto de datos y los parámetros del modelo en la memoria de la GPU
    B. En la GPU, luego tenemos mejor paralelismo para computar (muchas) multiplicaciones de matrices

- Además, PyTorch tiene diferenciación automática (más adelante)

- También, PyTorch implementa muchas funciones convenientes para el aprendizaje profundo (más adelante)

¡Cargar datos en la GPU es fácil!

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

```
In [23]: print(torch.cuda.is_available())
True

In [24]: b = b.to(torch.device('cuda:0'))
    ...: print(b)

tensor([1., 2., 3.], device='cuda:0')

In [25]: b = b.to(torch.device('cpu'))
    ...: print(b)
tensor([1., 2., 3.])
```

Cómo verificar tus dispositivos CUDA

- Si tienes CUDA instalado, deberías tener acceso a nvidia-smi
- Sin embargo, si estás usando un portátil, probablemente no tengas tarjetas gráficas compatibles con CUDA (mis portátiles no lo tienen)
- Hablaremos sobre computación en la nube con GPU más adelante...

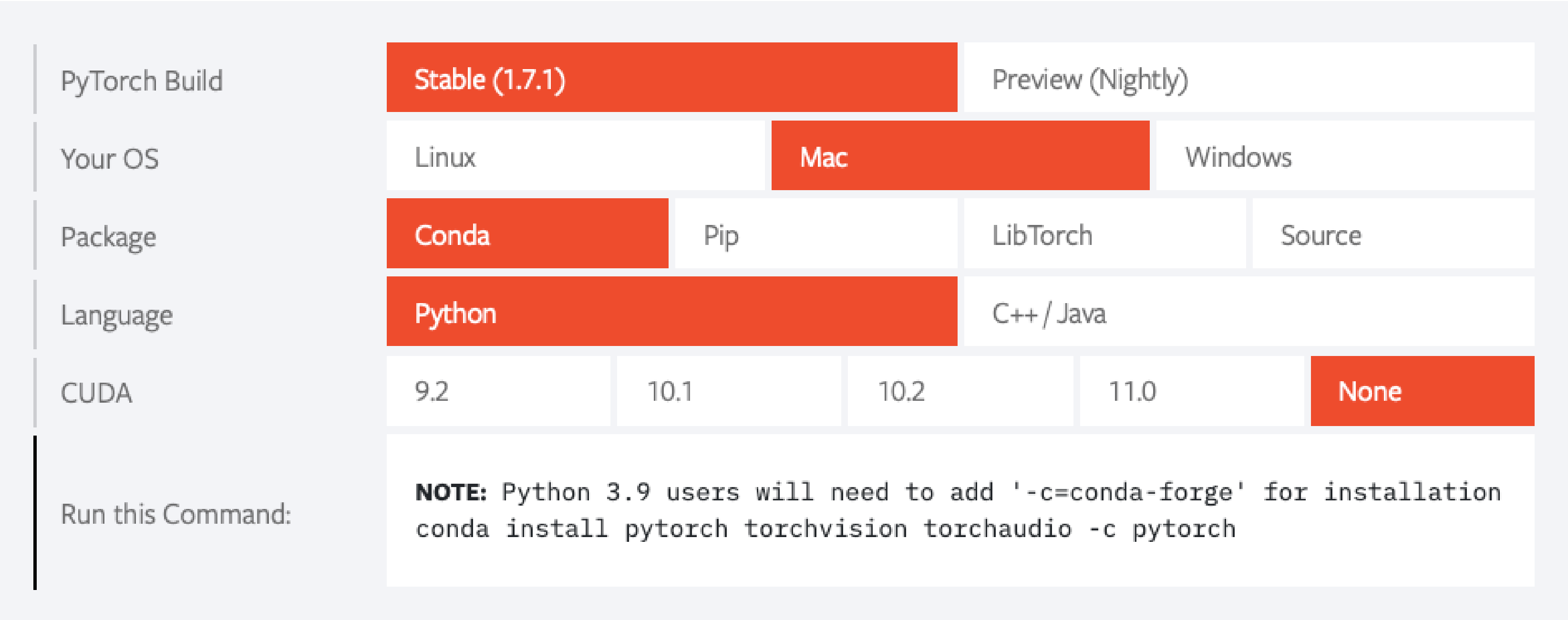
```
sraschka@gpu03:~$ nvidia-smi
Mon Feb  8 21:05:27 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.32.00    Driver Version: 455.32.00    CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  Off  | 00000000:1A:00.0 Off |                  N/A |
| 24%   37C    P0    71W / 250W |      0MiB / 11019MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Acerca de la instalación de PyTorch

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Si quieres instalar PyTorch después (tras la clase)...

- Si lo usas en un portátil, probablemente no tengas una GPU compatible con CUDA
- Se recomienda usar la versión para CPU en tu portátil (sin CUDA)
- Instalación en la nube con GPU más adelante...
- También, usa esta herramienta de selección desde https://pytorch.org (se recomienda conda):



Semántica de broadcasting:

Haciendo los cálculos vectoriales y matriciales más convenientes

Escuela de
Ciencias Aplicadas
e Ingeniería

1. Tensores en el aprendizaje profundo
2. Tensores y PyTorch
3. **Vectores, matrices y broadcasting**
4. Convenciones de notación para redes neuronales
5. Una capa totalmente conectada (lineal) en PyTorch

Vectores

¿Cómo llamamos a esto nuevamente en el contexto de las redes neuronales?

$$\mathbf{w}^\top \mathbf{x} + b = z \qquad \text{where} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

<u>Operaciones básicas con vectores</u>

- Suma (/resta)
- Productos internos (por ejemplo, producto punto)
- Multiplicación escalar

Los tensores de TensorFlow y PyTorch no son tensores reales

Universidad EAFIT — Escuela de Ciencias Aplicadas e Ingeniería

```
In [2]: a = torch.tensor([1, 2, 3])

In [3]: b = torch.tensor([4, 5, 6])

In [4]: a * b
Out[4]: tensor([ 4, 10, 18])

In [5]: torch.tensor([1, 2, 3]) + 1
Out[5]: tensor([2, 3, 4])
```

Aunque no son equivalentes a las definiciones matemáticas, ¡son muy útiles para computar!

(Estas “extensiones” también se utilizan comúnmente ahora en la notación matemática en la literatura de ciencias de la computación, ya que son bastante convenientes)

Matrices

Calculando la salida a partir de múltiples ejemplos de entrenamiento a la vez

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

$$\mathbf{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \dots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \dots & x_m^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \dots & x_m^{[n]} \end{bmatrix}$$

Dos oportunidades de paralelismo:

- multiplicar elementos para calcular el producto punto
- calcular múltiples productos punto

Calculando la salida a partir de múltiples ejemplos de entrenamiento a la vez

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Dos oportunidades de paralelismo:

- calcular el producto punto en paralelo
- calcular múltiples productos punto a la vez

$$\mathbf{X}\mathbf{w} + b = \mathbf{z}$$

donde

$$\mathrm{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \dots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \dots & x_m^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \dots & x_m^{[n]} \end{bmatrix}, \quad \mathrm{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} \mathbf{w}^\top \mathbf{x}^{[1]} + b \\ \mathbf{w}^\top \mathbf{x}^{[2]} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{[n]} + b \end{bmatrix} = \begin{bmatrix} z^{[1]} \\ z^{[2]} \\ \vdots \\ z^{[n]} \end{bmatrix}$$

(por eso w no es un "vector"
sino una matriz de m × 1)

Calculando la salida a partir de múltiples ejemplos de entrenamiento a la vez

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

$$\mathbf{X}\mathbf{w} + b = \mathbf{z}$$

(por eso w no es un "vector" sino una matriz de m × 1)

Pero NumPy y PyTorch no son muy exigentes con eso:

```
In [1]: import torch

In [2]: X = torch.arange(6).view(2, 3)

In [3]: X
Out[3]:
tensor([[0, 1, 2],
        [3, 4, 5]])
In [4]: w = torch.tensor([1, 2, 3])

In [5]: X.matmul(w)
Out[5]: tensor([ 8, 26])

In [6]: w = w.view(-1, 1)

In [7]: X.matmul(w)
Out[7]:
tensor([[ 8],
        [26]])
```

igual que reshape (razones históricas)

Calculando la salida a partir de múltiples ejemplos de entrenamiento a la vez

Escuela de Ciencias Aplicadas e Ingeniería

Dos oportunidades de paralelismo:

- calcular el producto punto en paralelo
- calcular múltiples productos punto a la vez

$$\mathbf{X}\mathbf{w} + b = \mathbf{z}$$

donde

$$\mathbf{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \dots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \dots & x_m^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \dots & x_m^{[n]} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

(por eso w no es un "vector" sino una matriz de m × 1)

$$\mathbf{z} = \begin{bmatrix} \mathbf{w}^\top \mathbf{x}^{[1]} + b \\ \mathbf{w}^\top \mathbf{x}^{[2]} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{[n]} + b \end{bmatrix} = \begin{bmatrix} z^{[1]} \\ z^{[2]} \\ \vdots \\ z^{[n]} \end{bmatrix}$$

¿Puedes detectar el error en esta diapositiva?

Calculando la salida a partir de múltiples ejemplos de entrenamiento a la vez

$\mathbf{X}\mathbf{w} + b = \mathbf{z}$

¿Puedes detectar el error en esta diapositiva?

Esto debería ser

$\mathbf{X}\mathbf{w} + \mathbf{1}_m b = \mathbf{z}$

pero ¡nosotros los investigadores en aprendizaje profundo somos perezosos! :)

Broadcasting

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- En PyTorch, funciona perfectamente.
- Esta característica (general) se llama "broadcasting"

```
In [4]: torch.tensor([1, 2, 3]) + 1
Out[4]: tensor([2, 3, 4])
```

```
In [5]: t = torch.tensor([[4, 5, 6], [7, 8, 9]])

In [6]: t
Out[6]:
tensor([[4, 5, 6],
        [7, 8, 9]])

In [7]: t + torch.tensor([1, 2, 3])
Out[7]:
tensor([[ 5,  7,  9],
        [ 8, 10, 12]])
```

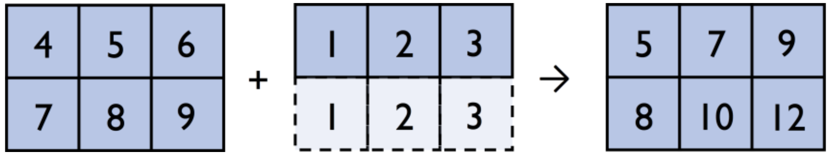
Broadcasting

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- En PyTorch, funciona perfectamente.
- Esta característica (general) se llama "broadcasting"

```
In [4]: torch.tensor([1, 2, 3]) + 1
Out[4]: tensor([2, 3, 4])
```


```
In [5]: t = torch.tensor([[4, 5, 6], [7, 8, 9]])

In [6]: t
Out[6]:
tensor([[4, 5, 6],
        [7, 8, 9]])

In [7]: t + torch.tensor([1, 2, 3])
Out[7]:
tensor([[ 5,  7,  9],
        [ 8, 10, 12]])
```



Se agregan dimensiones implícitas, los elementos se duplican implícitamente.

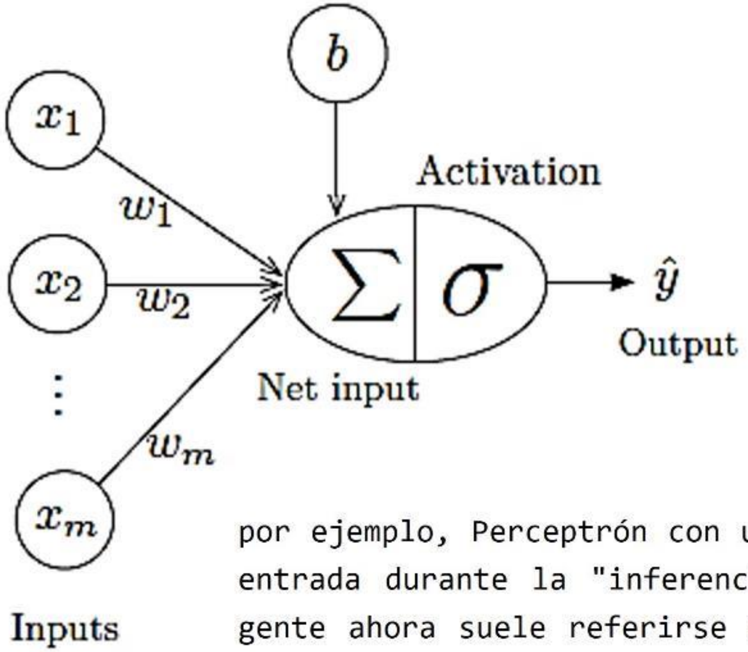
Convenciones Notacionales de Álgebra Lineal en Aprendizaje Profundo

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1. Tensores en el aprendizaje profundo
2. Tensores y PyTorch
3. Vectores, matrices y broadcasting
4. **Convenciones de notación para redes neuronales**
5. Una capa totalmente conectada (lineal) en PyTorch

Conexiones que ya hemos visto...

$$\mathbf{x}^\top \mathbf{w} + b = z$$

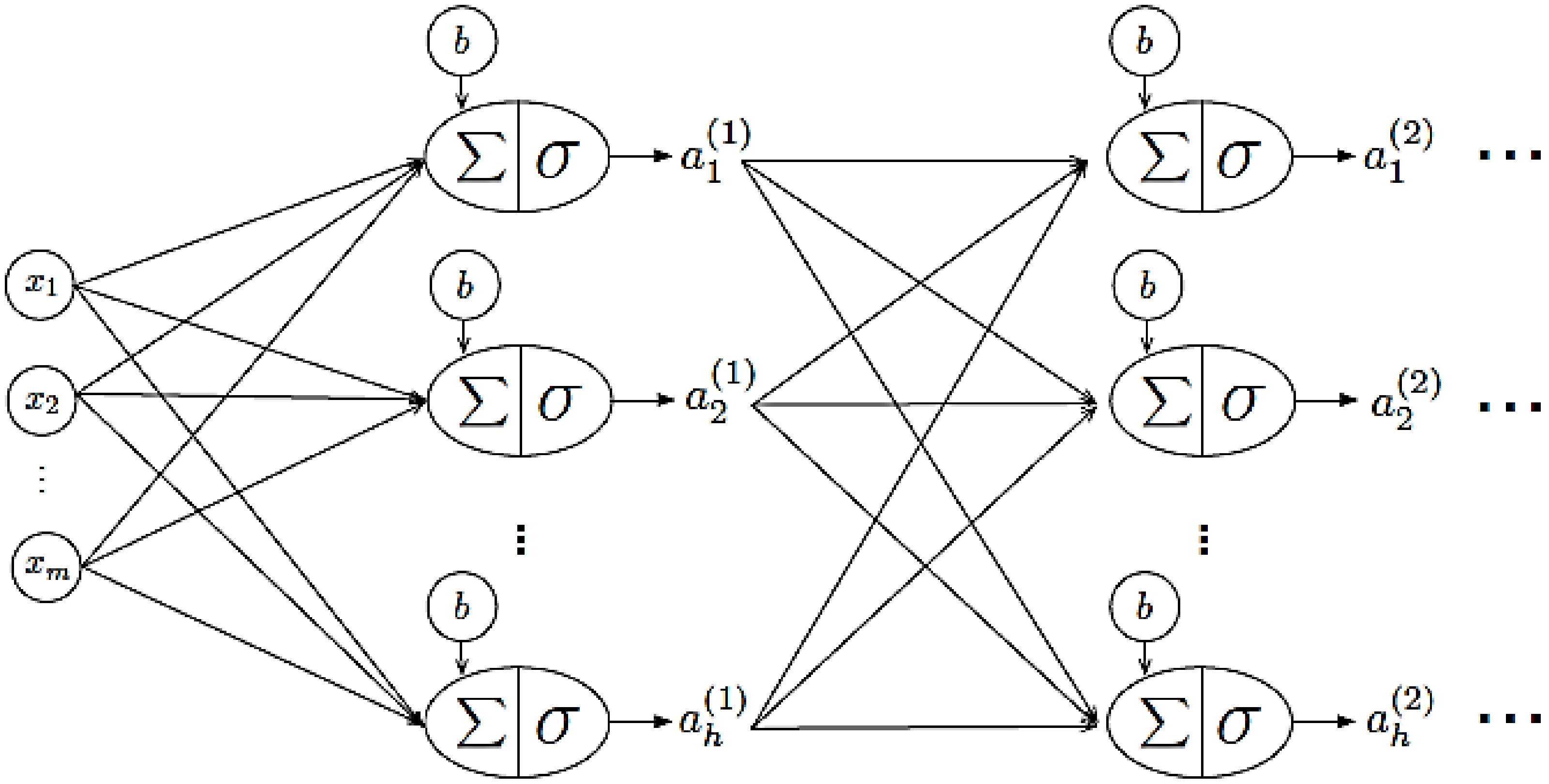


por ejemplo, Perceptrón con un ejemplo de entrenamiento como entrada durante la "inferencia"(en aprendizaje profundo, la gente ahora suele referirse a predecir la variable objetivo como "inferencia")

Si tenemos n ejemplos de entrenamiento, $\mathbf{X} \in \mathbb{R}^{n \times m}$, $\mathbf{z} \in \mathbb{R}^{n \times 1}$

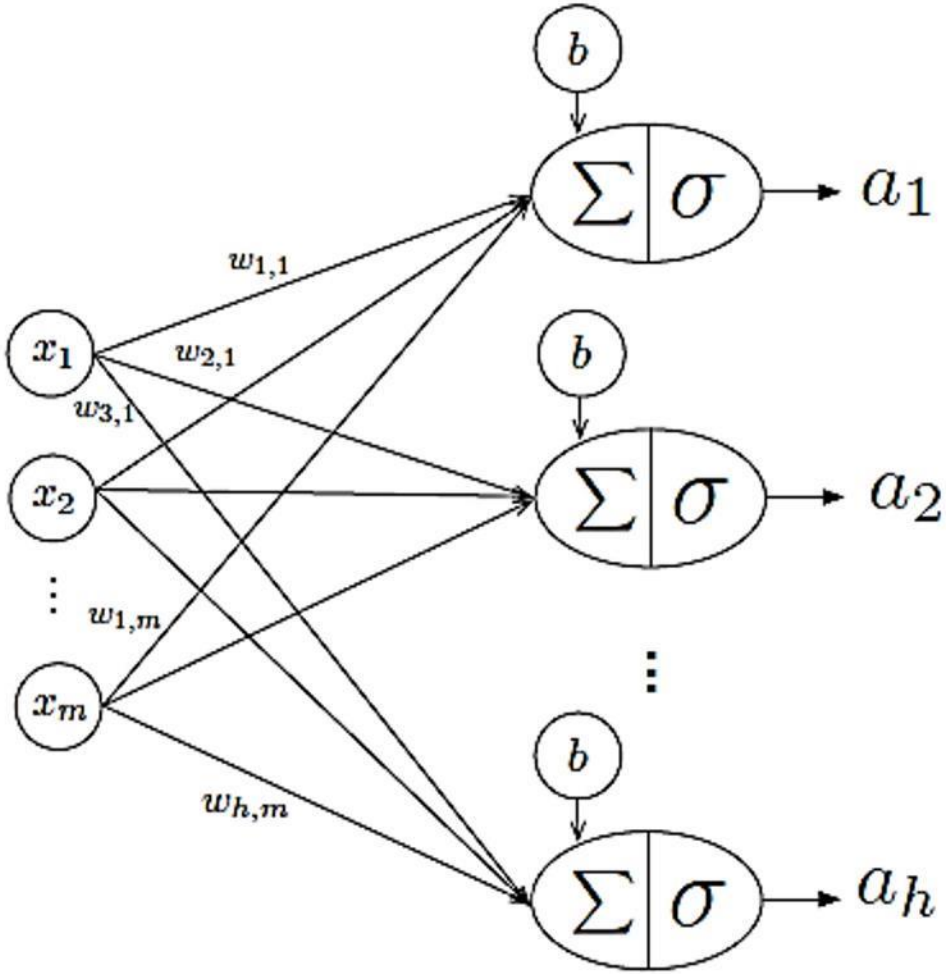
$$\mathbf{X}\mathbf{w} + b = \mathbf{z}$$

Conexiones que encontraremos más adelante...



Una capa totalmente conectada

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



$$\text{where} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{h,2} & \cdots & w_{h,m} \end{bmatrix}$$

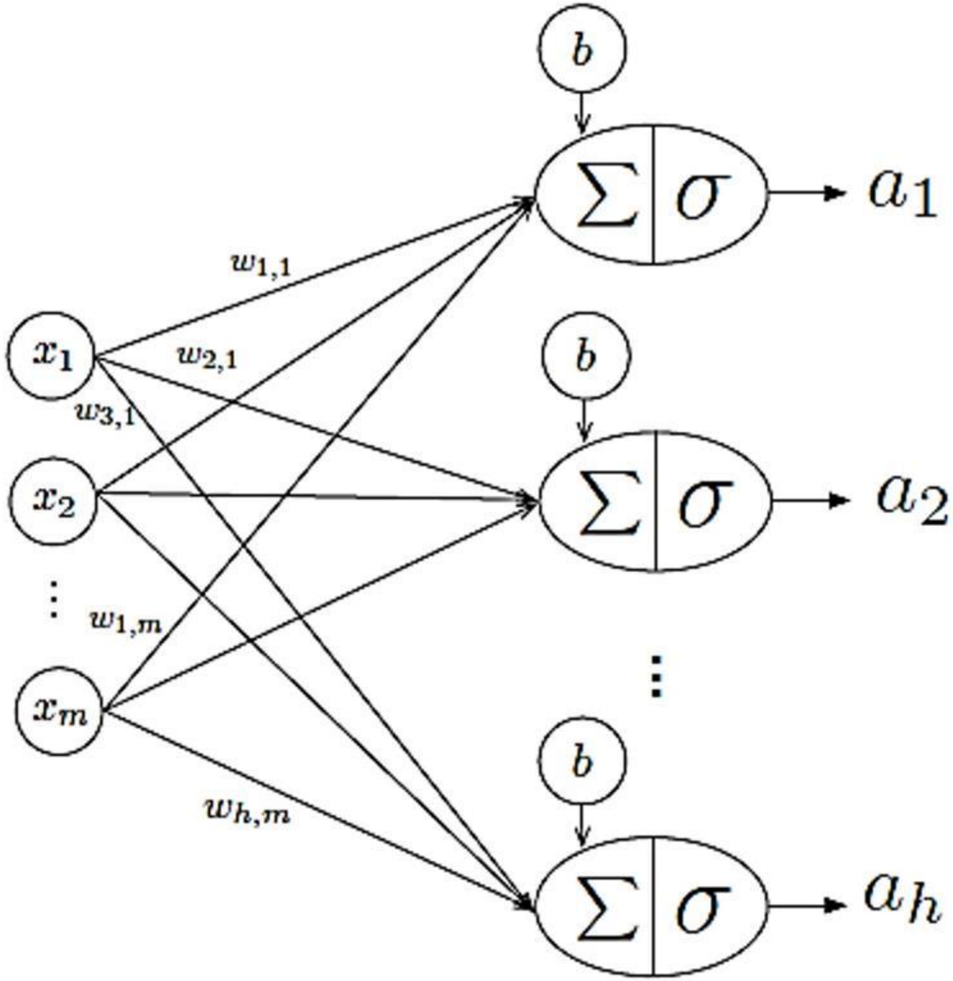
Activaciones de capa para 1 ejemplo de entrenamiento

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{a}$$

$$\mathbf{a} \in \mathbb{R}^{h \times 1}$$

nota que *wi*,w se refiere al peso que conecta laentrada j-ésima con la salida i-ésima.

Una capa totalmente conectada



Activaciones de capa para n ejemplos de entrenamiento

$$\sigma([\mathbf{W}\mathbf{X}^\top + \mathbf{b}]^\top) = \mathbf{A}$$

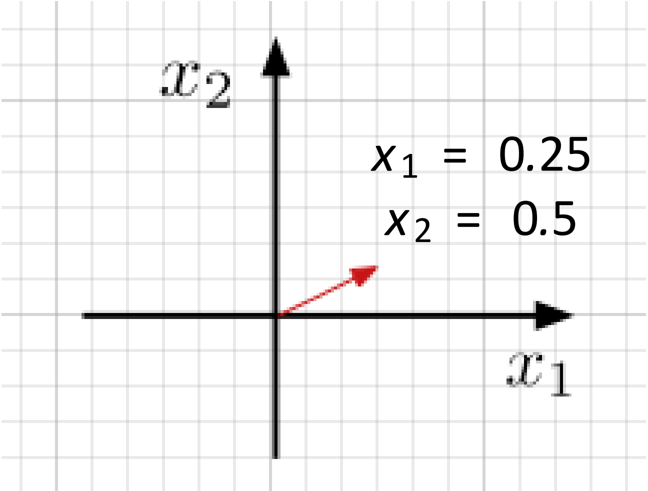
$$\mathbf{A} \in \mathbb{R}^{n \times h}$$

Los libros de texto de aprendizaje automático usualmente representan los ejemplos de entrenamiento sobre las columnas, y las características sobre las filas (en lugar de usar la "matriz de diseño") – en ese caso, podríamos omitir la transpuesta.

¿Pero por qué la notación Wx es intuitiva?

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Matriz de transformación



¿Pero por qué la notación Wx es intuitiva?

escala la coordenada x

mueve y en la dirección de x

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ d \end{bmatrix} + y \begin{bmatrix} b \\ c \end{bmatrix}$$

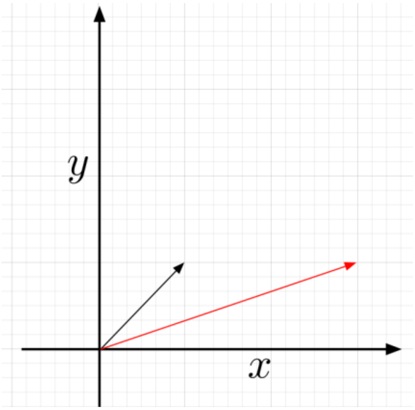
mueve x en la dirección de y

escala la coordenada y

¿Pero por qué la notación Wx es intuitiva?

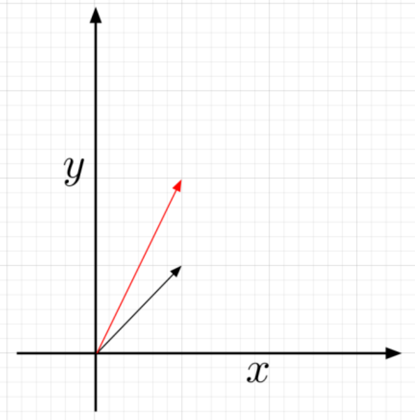
Estiramiento del eje x por un factor de 3

$$\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x \\ y \end{bmatrix}$$



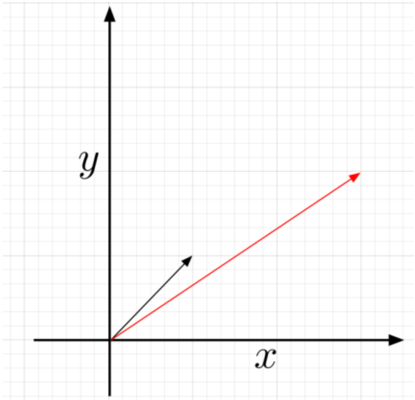
Estiramiento del eje y por un factor de 2

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ 2y \end{bmatrix}$$



Estiramiento del eje x por un factor de 3 y del eje y por un factor de 2

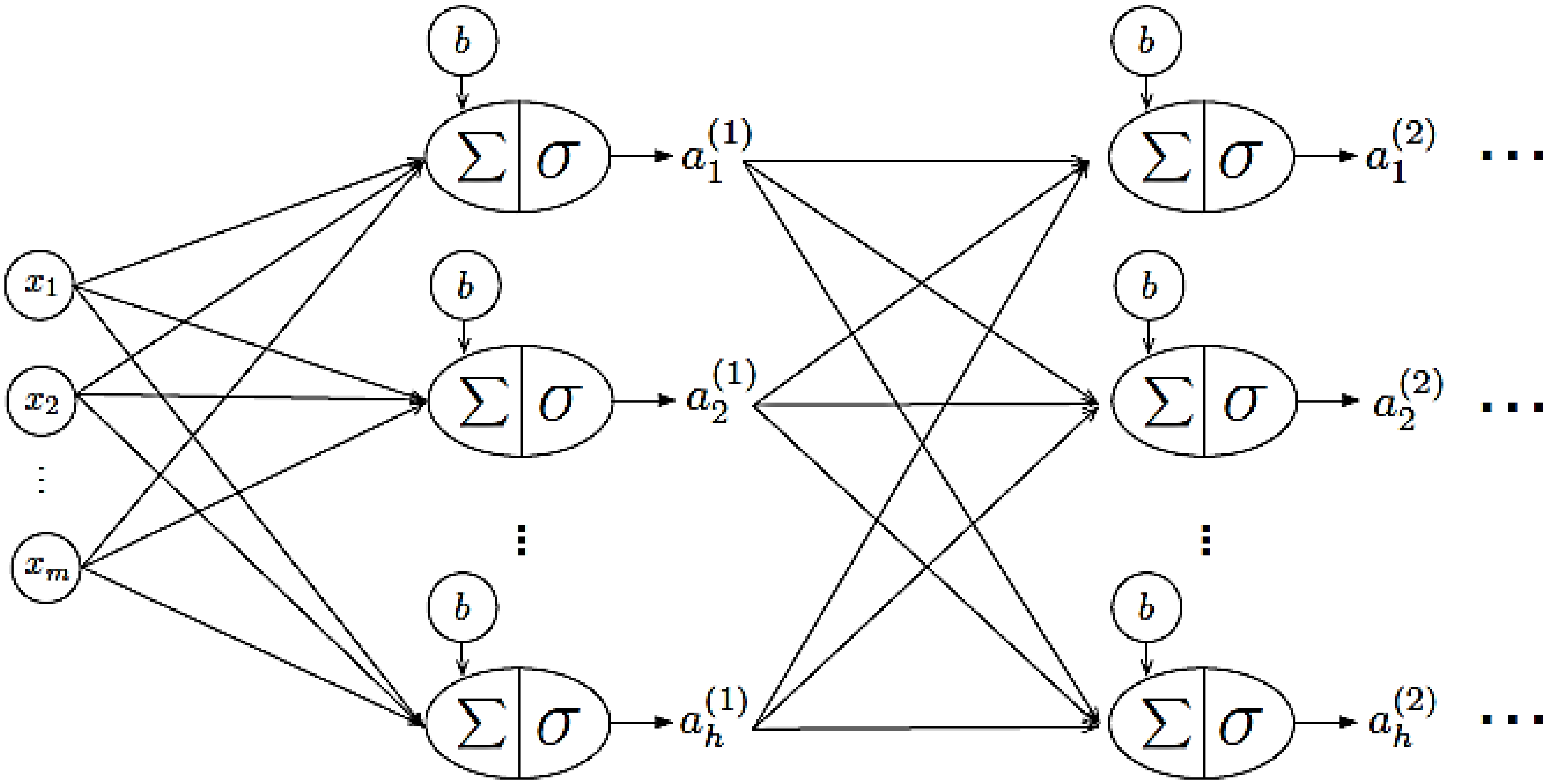
$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x \\ 2y \end{bmatrix}$$



Una capa totalmente conectada (lineal) en PyTorch

1. Tensores en el aprendizaje profundo
2. Tensores y PyTorch
3. Vectores, matrices y broadcasting
4. Convenciones de notación para redes neuronales
5. **Una capa totalmente conectada (lineal) en PyTorch**

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Capa totalmente conectada en PyTorch

```
[1]: import torch
```

```
[2]: X = torch.arange(50, dtype=torch.float).view(10, 5)
     # .view() and .reshape() are equivalent
     X
```

```
[2]: tensor([[ 0.,  1.,  2.,  3.,  4.],
             [ 5.,  6.,  7.,  8.,  9.],
             [10., 11., 12., 13., 14.],
             [15., 16., 17., 18., 19.],
             [20., 21., 22., 23., 24.],
             [25., 26., 27., 28., 29.],
             [30., 31., 32., 33., 34.],
             [35., 36., 37., 38., 39.],
             [40., 41., 42., 43., 44.],
             [45., 46., 47., 48., 49.]])
```

```
[3]: fc_layer = torch.nn.Linear(in_features=5,
                                out_features=3)
```

```
[4]: fc_layer.weight
```

```
[4]: Parameter containing:
     tensor([[-0.1706,  0.1684,  0.3509,  0.1649,  0.1903],
             [-0.1356,  0.0663, -0.4357,  0.2710,  0.1179],
             [-0.0736,  0.0413, -0.0186,  0.4032,  0.0992]], requires_grad=True)
```

```
[5]: fc_layer.bias
```

```
[5]: Parameter containing:
     tensor([-0.2552,  0.3918,  0.2693], requires_grad=True)
```

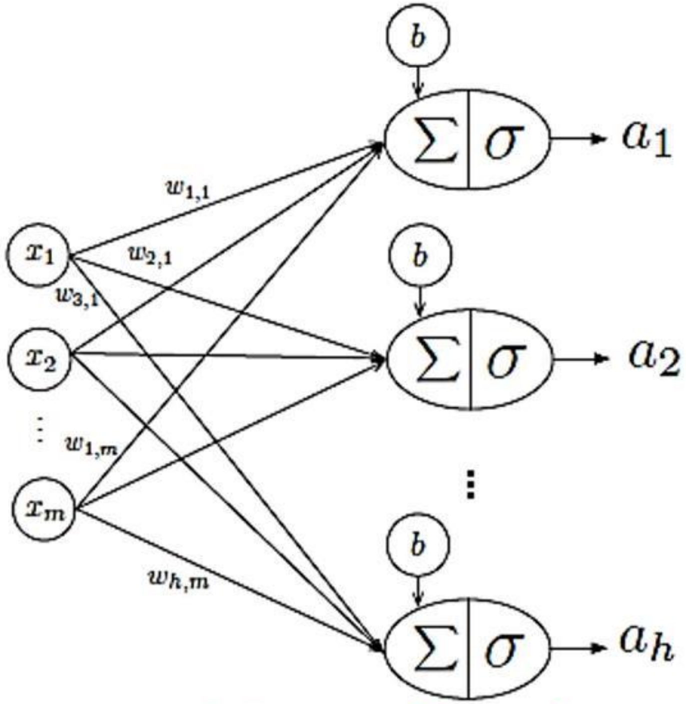
Capa totalmente conectada en PyTorch

Escuela de Ciencias Aplicadas e Ingeniería

```
[6]: print('X dim:', X.size())
print('W dim:', fc_layer.weight.size())
print('b dim:', fc_layer.bias.size())
# .size() is equivalent to .shape
A = fc_layer(X)
print('A:', A)
print('A dim:', A.size())
```

```
X dim: torch.Size([10, 5])
W dim: torch.Size([3, 5])
b dim: torch.Size([3])
A: tensor([[ 1.2004,  2.3291,  2.0036],
        [ 4.5367,  7.7858,  5.4519],
        [ 7.8730, 13.2424,  8.9003],
        [11.2093, 18.6991, 12.3486],
        [14.5457, 24.1557, 15.7970],
        [17.8820, 29.6123, 19.2453],
        [21.2183, 35.0690, 22.6937],
        [24.5546, 40.5256, 26.1420],
        [27.8910, 45.9823, 29.5904],
        [31.2273, 51.4389, 33.0387]], grad_fn=<ThAddmmBackward>)
A dim: torch.Size([10, 3])
```

Basado en PyTorch, tenemos otra convención



Nota que *wi,j* se refiere al peso que conecta la entrada j-ésima con la salida i-ésima.

...

where $W = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{h,2} & \dots & w_{h,m} \end{bmatrix}$

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 \dots & x_m \end{bmatrix}$$

Activaciones de la capa para 1 ejemplo de entrenamiento

$$\sigma(\mathbf{x}\mathbf{W}^\top + \mathbf{b}) = \mathbf{a}$$

$$\mathbf{a} \in \mathbb{R}^{1\times h}$$

Activaciones de la capa para n ejemplos de entrenamiento

$$\sigma(\mathbf{X}\mathbf{W}^\top + \mathbf{b}) = \mathbf{A}$$

$$\mathbf{W}^\top \in \mathbb{R}^{m\times h}$$

$$\mathbf{A} \in \mathbb{R}^{n\times h}$$

Puedes encontrar el código fuente aquí
https://github.com/pytorch/pytorch/blob/18edd3ab0828acaa81dc052dba8644c874dc62db/torch/nn/functional.py#L1368

Conclusión

- Piensa siempre en cómo se calculan los productos punto al escribir e implementar la multiplicación de matrices.
- La intuición teórica y la convención no siempre coinciden con la conveniencia práctica (al programar).
- Al cambiar entre la teoría y el código, estas reglas pueden ser útiles:

$$\mathbf{AB} = (\mathbf{B}^\top \mathbf{A}^\top)^\top$$

$$(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$$

Resumen: Tradicional vs PyTorch

(La matriz de transformación idealmente debería ir siempre al frente)

where $\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{h,2} & \cdots & w_{h,m} \end{bmatrix}$

ten en cuenta que wi,j se refiere al peso que conecta la entrada j con la salida i..

Activaciones de capa para 1 ejemplo de entrenamiento

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{a} \ , \ \mathbf{a} \in \mathbb{R}^{h \times 1} \quad \text{with } \ \mathbf{x} \in \mathbb{R}^{m \times 1}$$

$$\Leftrightarrow \sigma([\mathbf{x}^\top \mathbf{W}^\top]^\top + \mathbf{b}) = \mathbf{a} \quad \text{with } \ \mathbf{x} \in \mathbb{R}^{m \times 1}$$

$$\Leftrightarrow \sigma([\mathbf{x}\mathbf{W}^\top] + \mathbf{b}) = \mathbf{a} \quad \text{with } \ \mathbf{x} \in \mathbb{R}^{1 \times m} \ \text{(PyTorch)}$$

Activaciones de capa para n ejemplos de entrenamiento

$$\sigma([\mathbf{W}\mathbf{X}^\top]^\top + \mathbf{b}) = \mathbf{A} \ , \ \mathbf{A} \in \mathbb{R}^{n \times h} \quad \text{with } \ \mathbf{X} \in \mathbb{R}^{n \times m}$$

$$\Leftrightarrow \sigma([\mathbf{X}\mathbf{W}^\top] + \mathbf{b}) = \mathbf{A} \quad \text{with } \ \mathbf{X} \in \mathbb{R}^{n \times m}$$

Ejercicio / Experimento de tarea no calificada

• Revisita nuestro código del perceptrón en NumPy:
https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L03-perceptron/code/perceptron-numpy.ipynb

1. Sin ejecutar el código, ¿puedes decir si el perceptrón podría predecir las etiquetas de clase si alimentamos un arreglo de múltiples ejemplos de entrenamiento a la vez (es decir, mediante su método forward)?

   - ¿Sí? ¿Por qué?
   - ¿No? ¿Qué cambio necesitarías hacer?

2. Ejecuta el código para verificar tu intuición.

3. ¿Y qué hay del método train? ¿Podemos tener paralelismo a través de multiplicación matricial sin afectar la regla de aprendizaje del perceptrón?