Dive into Deep Learning

Chapter 2. Preliminaries

Wayne L, Oct 4, 2020

2.1 Data manipulation

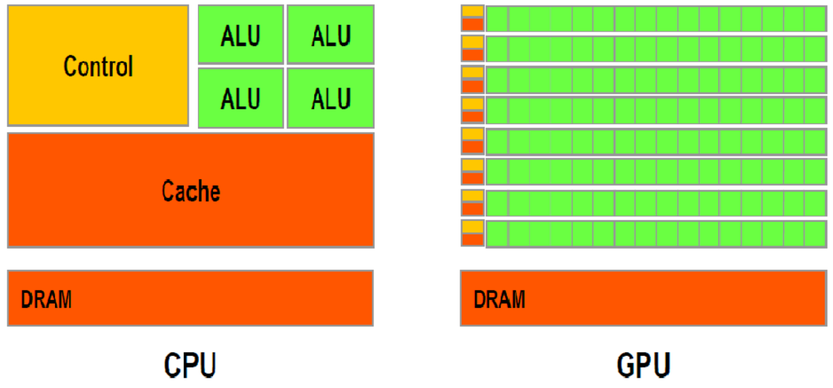
Introduction

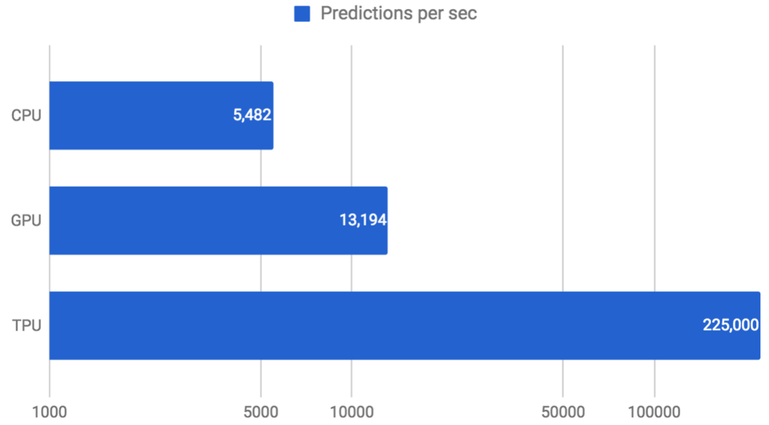
- extracting information from data
- learning the practical skills for storing, manipulating and preprocessing data
- requires working with large datasets (row: examples, columns: attributes)
- Linear algebra is good for handling the tabular data
- all about optimization (automatic differentiation)
- making predictions: find the likely value of some unknown attribute, given the information that we observe

2.1 Data manipulation

Introduction

- Important things: (i) acquire (ii) process
- useful framework and library: mxnet PyTorch NumPy
- GPU is well-supported to accelerate the computation
- the tensor class supports automatic differentiation

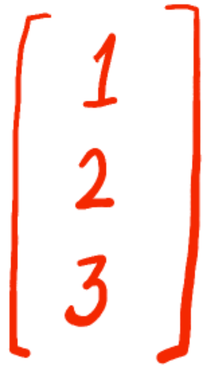




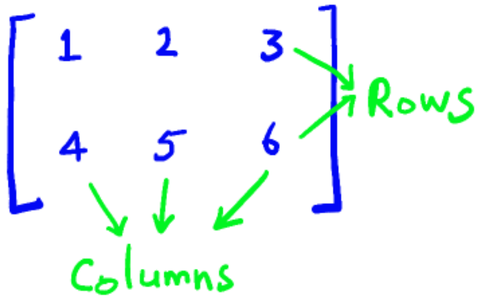
2.1 Data manipulation

2.1.1 Getting started

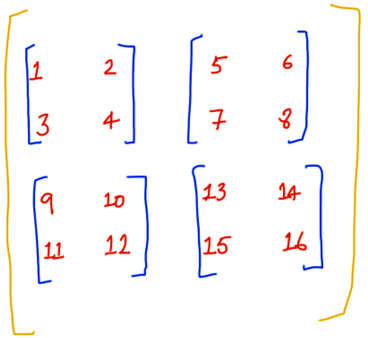
Scalar
(0-axis)



Vector
(1-axis)



Matrix
(2-axes)



Tensor
(n-axes)

- arange, shape, size, reshape, …
- ones, zeros, …

2.1 Data manipulation

2.1.1 Getting started

- Shape: the size of the dimensions of an N-dimensional array
- Size: the amount (or count) of elements that are contained in the array
- Reshaping by manually specifying every dimension is unnecessary

```
[32] import numpy as np
```

```
[33] x = np.arange(12)
     x
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
[34] x.shape
```

```
(12,)
```

```
x.size
```

```
12
```

```
[37] X = x.reshape(3, 4)
     X
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
[39] X_ = x.reshape(3,-1)
     X_
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
[40] X_ = x.reshape(-1, 4)
     X_
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

2.1 Data manipulation

2.1.1 Getting started

- assign initial value
  - zeros
  - ones
  - some other constants
  - randomly sampled from a specific distribution

```
[41] np.zeros((2,3,4))
```

```
array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```

```
[42] np.random.normal(0, 1, size=(3,4))
```

```
array([[-0.05481032,  0.0767631 ,  0.0338954 ,  0.91264142],
       [ 1.18868263, -0.86146151,  1.10383142, -0.92105561],
       [-2.0089984 , -0.48289094, -1.0383175 ,  1.11282547]])
```

```
[43] np.array([[2,3,4,1],[6,7,8,4],[3,2,1,2]])
```

```
array([[2, 3, 4, 1],
       [6, 7, 8, 4],
       [3, 2, 1, 2]])
```

2.1 Data manipulation

2.1.2 Operations

- elementwise
  - +, -, *, / and **
  - exponential
  - concatenate
  - logical statement (==, <, >)
  - sum

```
[13] x = np.array([1,2,4,8])
     y = np.array([2,2,2,2])
     x + y, x - y, x * y, x / y, x ** y
```

```
(array([ 3,  4,  6, 10]),
 array([-1,  0,  2,  6]),
 array([ 2,  4,  8, 16]),
 array([0.5, 1. , 2. , 4. ]),
 array([ 1,  4, 16, 64]))
```

```
[14] np.exp(x)
```

```
array([2.71828183e+00, 7.38905610e+00, 5.45981500e+01, 2.98095799e+03])
```

```
[15] X = np.arange(12).reshape(3, 4)
     Y = np.array([[2,1,4,3],[1,2,3,4],[4,3,2,1]])
     np.concatenate([X, Y], axis=0)
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 2,  1,  4,  3],
       [ 1,  2,  3,  4],
       [ 4,  3,  2,  1]])
```

```
[16] np.concatenate([X, Y], axis=1)
```

```
array([[ 0,  1,  2,  3,  2,  1,  4,  3],
       [ 4,  5,  6,  7,  1,  2,  3,  4],
       [ 8,  9, 10, 11,  4,  3,  2,  1]])
```

```
[17] X == Y
```

```
array([[False,  True, False,  True],
       [False, False, False, False],
       [False, False, False, False]])
```

```
[18] X.sum()
```

```
66
```

2.1 Data manipulation

2.1.3 Broadcasting Mechanism

- When shapes differ, we can still perform elementwise operations by invoking the *broadcasting mechanism*

```
[19] a = np.arange(3).reshape(3, 1)
     b = np.arange(2).reshape(1, 2)
     a, b
```

```
(array([[0],
        [1],
        [2]]), array([[0, 1]]))
```

```
[20] a + b
```

```
array([[0, 1],
       [1, 2],
       [2, 3]])
```

2.1 Data manipulation

2.1.4 Indexing and Slicing

- [-1] selects the last element
- [1:3] selects the second and the third elements
- overwrite elements by specifying indices
- [0:2, :] access the first and second rows, where : takes all the elements along axis 1.

```
[21] X[-1], X[1:3]
```

```
(array([ 8,  9, 10, 11]), array([[ 4,  5,  6,  7],
        [ 8,  9, 10, 11]]))
```

```
[26] X[1, 2] = 9
     X
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  9,  7],
       [ 8,  9, 10, 11]])
```

```
[27] X[0:2, :] = 12
     X
```

```
array([[12, 12, 12, 12],
       [12, 12, 12, 12],
       [ 8,  9, 10, 11]])
```

2.1 Data manipulation

2.1.5 Saving Memory

- Running operations can cause new memory to be allocated unnecessarily all the time
- In ML, we might have hundreds of MB of parameters and update all of them multiple times per second.
- We will want to perform these updates *in place*
  (i) X[:] = X + Y
  (ii) X += Y
  easily reduce the memory overhead

```
[29] before = id(Y)
     Y = Y + X
     id(Y) == before
```

```
False
```

```
[33] Z = np.zeros_like(Y)
     print('id(Z):', id(Z))
     Z[:] = X + Y
     print('id(Z):', id(Z))
```

```
id(Z): 140454541982176
id(Z): 140454541982176
```

```
[32] before = id(X)
     X += Y
     id(X) == before
```

```
True
```

2.1 Data manipulation

2.1.6 Conversion to Other Python Objects

- Converting to a Numpy tensor, or vice versa, is easy
- The converted result doesn’t share memory

2.2 Data Preprocessing

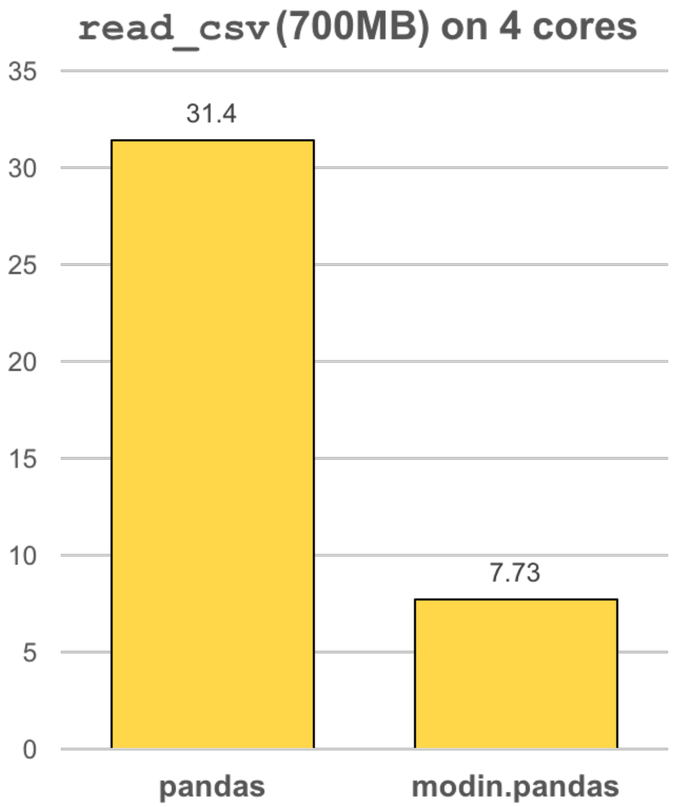
2.2.1 Reading the Dataset

- dataset that is stored in a csv or tsv
- pandas package is commonly used
- Link: https://github.com/modin-project/modin

```
# If pandas is not installed, just uncomment the following line:
# !pip install pandas
import pandas as pd

data = pd.read_csv(data_file)
print(data)
```

```
   NumRooms Alley   Price
0       NaN  Pave  127500
1       2.0   NaN  106000
2       4.0   NaN  178100
3       NaN   NaN  140000
```



2.2 Data Preprocessing

2.2.2 Handling Missing Data

- Note that “NaN” entries are missing values.
- *Imputation* replaces missing values with substituted ones
- *Deletion* ignores missing values

```
inputs, outputs = data.iloc[:, 0:2], data.iloc[:, 2]
inputs = inputs.fillna(inputs.mean())
print(inputs)
```

```
   NumRooms Alley
0       3.0  Pave
1       2.0   NaN
2       4.0   NaN
3       3.0   NaN
```

2.3 Linear Algebra

2.3.1 Scalars

- $c = \frac{5}{9}(f - 32)$

- 5, 9 and -32 are scalar values and the placeholders c and f are called variables and they represent unknown scalar values

- Represented by a tensor with just one element

```
[6]  import tensorflow.compat.v1 as tf

     with tf.Session() as session:
       f = tf.placeholder(tf.float32, name='f')
       c = 5/9 * (f - 32)
       print(session.run(c, feed_dict={f: 67.0}))
```

```
19.444445
```

```
[7]  import tensorflow.compat.v1 as tf

     with tf.Session() as session:
       x = tf.placeholder(tf.float32, [None, 4])
       y = x * 10 + 1
       data = [[12, 2, 0, -2], [14, 4, 1, -5]]
       print(session.run(y, feed_dict={x: data}))
```

```
[[121.  21.   1. -19.]
 [141.  41.  11. -49.]]
```

2.3 Linear Algebra

2.3.2 Vectors

- List of scalar values
- If we were studying the risk of heart attacks, we might represent each patient by a vector whose components capture their most recent vital signs, cholesterol levels, minutes of exercise per day, etc.
- In general tensors can have arbitrary lengths, subject to the memory limits of your machine

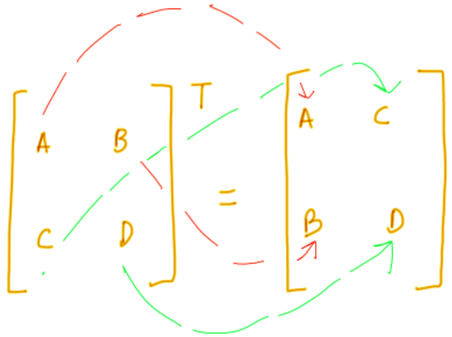
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

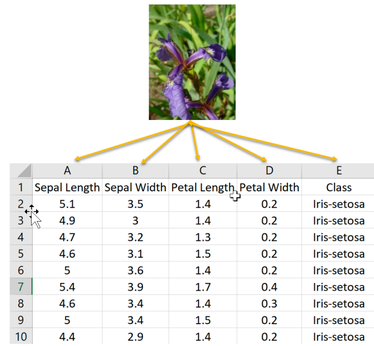
2.3 Linear Algebra

2.3.3 Matrices

- List of vectors
- Transpose: exchange a matrix's rows and columns
- Useful data structures: they allow us to organize data that have different modalities of variation.
- Rows correspond to data examples, while columns to different attributes

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$



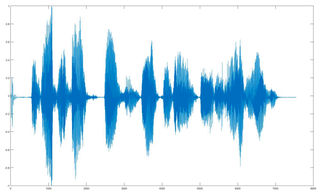


	A	B	C	D	E
1	Sepal Length	Sepal Width	Petal Length	Petal Width	Class
2	5.1	3.5	1.4	0.2	Iris-setosa
3	4.9	3	1.4	0.2	Iris-setosa
4	4.7	3.2	1.3	0.2	Iris-setosa
5	4.6	3.1	1.5	0.2	Iris-setosa
6	5	3.6	1.4	0.2	Iris-setosa
7	5.4	3.9	1.7	0.4	Iris-setosa
8	4.6	3.4	1.4	0.3	Iris-setosa
9	5	3.4	1.5	0.2	Iris-setosa
10	4.4	2.9	1.4	0.2	Iris-setosa

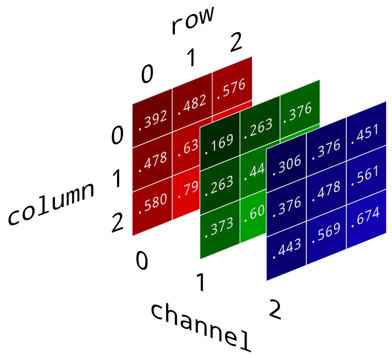
2.3 Linear Algebra

2.3.4 Tensors

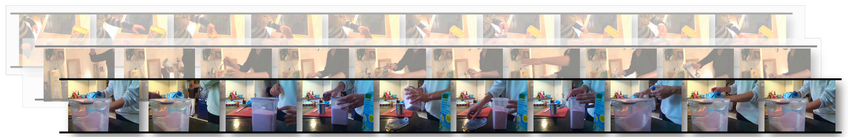
- *n*-dimensional arrays with an arbitrary number of axes



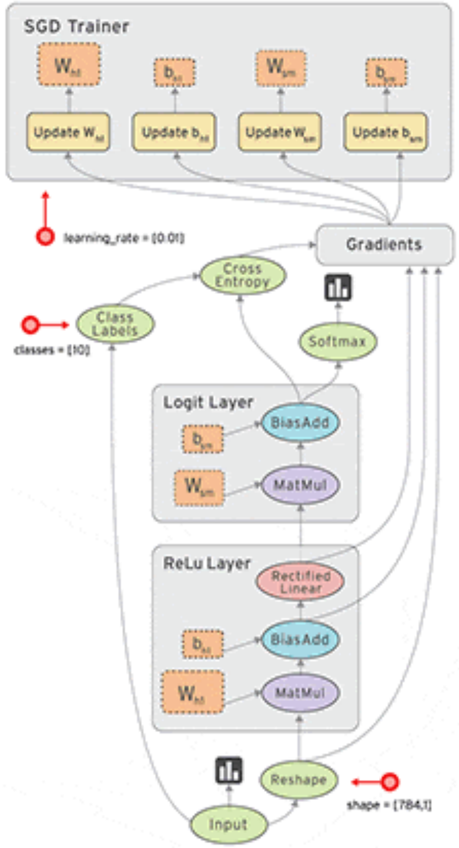
audio



image



video



2.3 Linear Algebra

2.3.5 Basic Properties of Tensor Arithmetic

- Multiplying or adding a tensor by a scalar doesn't change the shape of the tensor

```
[39] a = 2
     X = np.arange(24).reshape(2,3,-1)
```

```
[40] a+X
```

```
array([[[ 2,  3,  4,  5],
        [ 6,  7,  8,  9],
        [10, 11, 12, 13]],

       [[14, 15, 16, 17],
        [18, 19, 20, 21],
        [22, 23, 24, 25]]])
```

```
[41] (a*X).shape
```

```
(2, 3, 4)
```

2.3 Linear Algebra

2.3.6 Reduction

- similar to *group by*
- Non-reduction sum
  - keepdims = *True*

```
[42] A = np.arange(20).reshape(5,-1)
```

```
[46] A.sum(), A.shape
```

```
(190, (5, 4))
```

```
[44] A.sum(axis=0), A.sum(axis=0).shape
```

```
(array([40, 45, 50, 55]), (4,))
```

```
[45] A.sum(axis=1), A.sum(axis=1).shape
```

```
(array([ 6, 22, 38, 54, 70]), (5,))
```

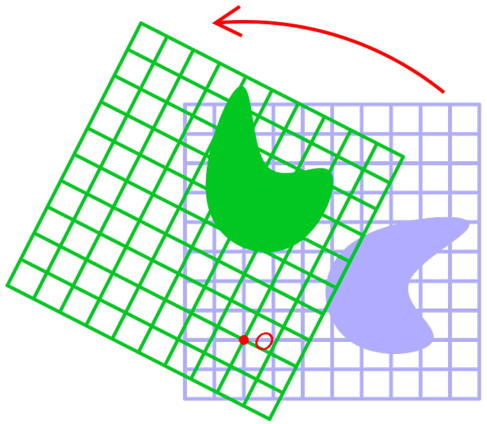
2.3 Linear Algebra

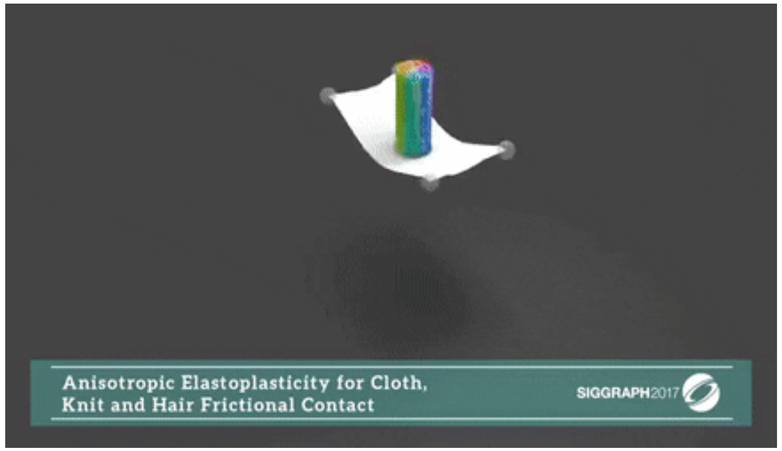
2.3.7 Dot Products (Matrix multiplication)

- using various fields in computer science

$$\begin{matrix} & \vec{b_1} & \vec{b_2} \\ & \downarrow & \downarrow \\ \vec{a_1}\rightarrow & & \\ \vec{a_2}\rightarrow & & \end{matrix} \begin{bmatrix} 1 & 7 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \vec{a_1} \cdot \vec{b_1} & \vec{a_1} \cdot \vec{b_2} \\ \vec{a_2} \cdot \vec{b_1} & \vec{a_2} \cdot \vec{b_2} \end{bmatrix}$$

$A \qquad B \qquad C$

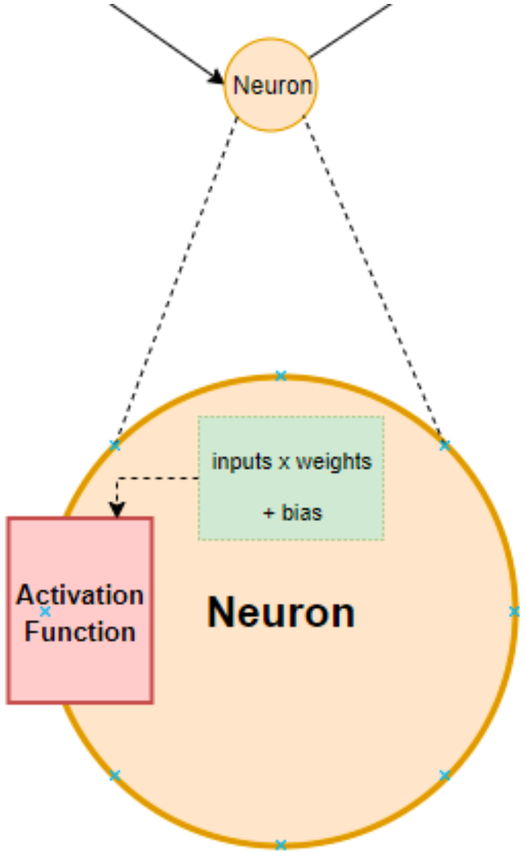




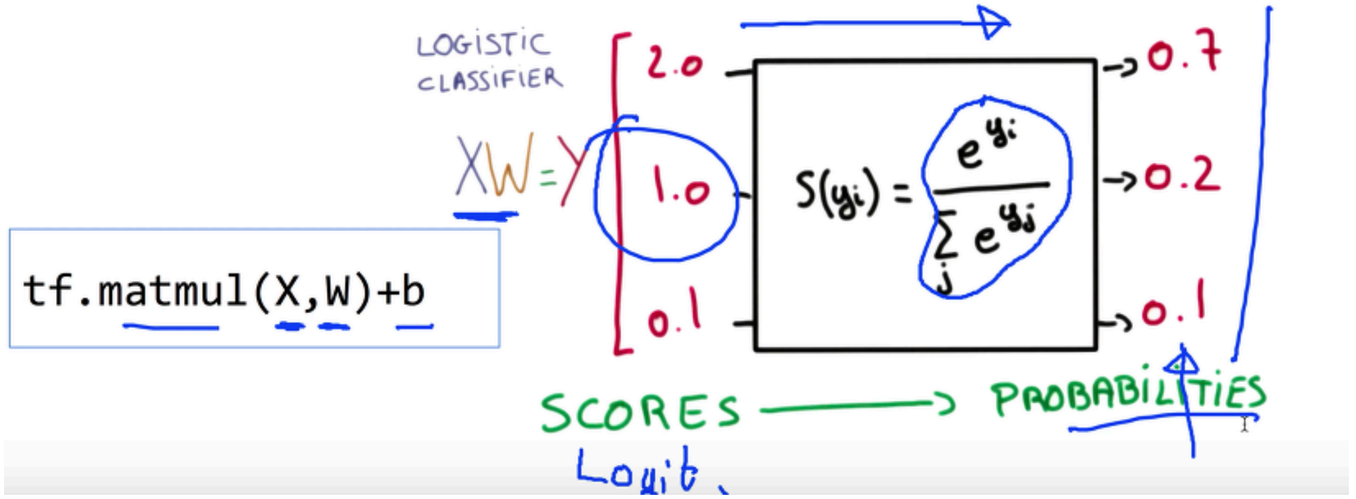
2.3 Linear Algebra

2.3.7 Dot Products (Matrix multiplication)

- Also, Deep-learning




```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```




```
tf.matmul(X,W)+b
```

2.3 Linear Algebra

2.3.10 Norms

- the norm of a vector tells us how big a vector is
- not dimensionality but magnitude of the components
- function f that maps a vector to a scalar, satisfying a handful of properties
  (i) $f(\alpha X) = |\alpha| f(X)$
  (ii) $f(X + Y) \leq f(X) + f(Y)$
  (iii) $f(X) \geq 0$

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

```
[10] u = np.array([3, -4])
     np.linalg.norm(u)
```

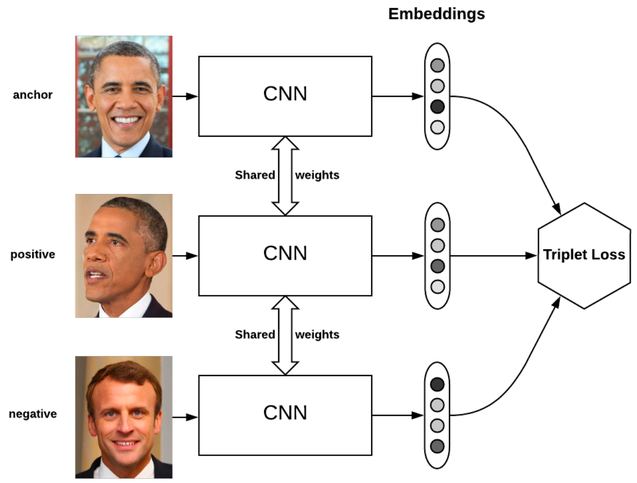
```
5.0
```

L_2 norm: Euclidean distance

2.3 Linear Algebra

2.3.10 Norms

- In deep learning, we are often trying to solve optimization problems
  - ***maximize*** the probability assigned to observed data
  - ***minimize*** the distance between predictions and the ground-truth observations



triplet loss

L1 Regularization

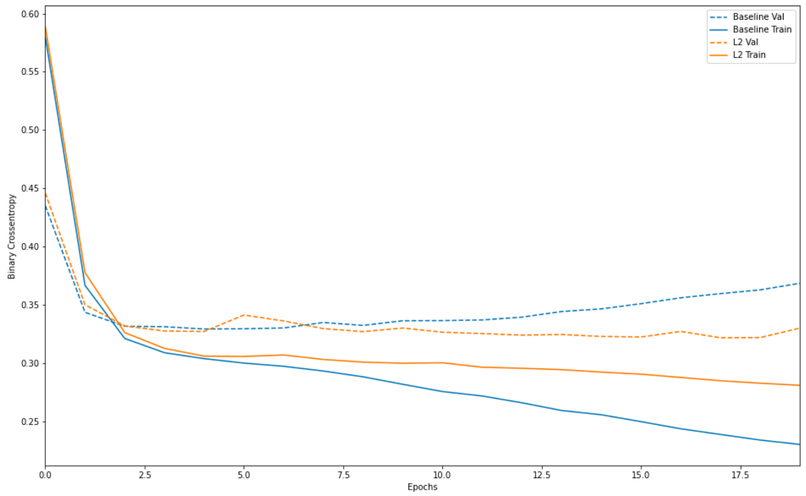
$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term



Regularization