Introduction to Deep Learning

October 1, 2021

Who am I?

Aaron Chang

Projects Director

aaron.chang@aisociety.io

ais

intro.nasuno.me

Agenda

- What is Deep Learning?
- Why Learn Deep Learning?
- Data
- Matrices with NumPy
- Manipulating Data with Pandas

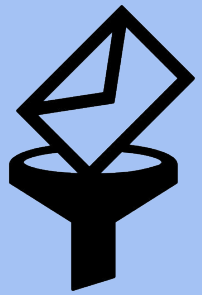
What is Deep Learning?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

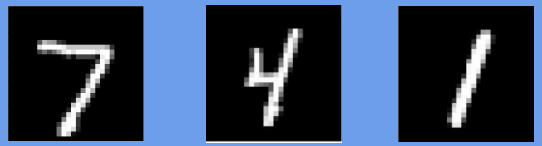
Machine Learning

Ability to learn without explicitly being programmed



Deep Learning

Extract patterns from data using neural networks



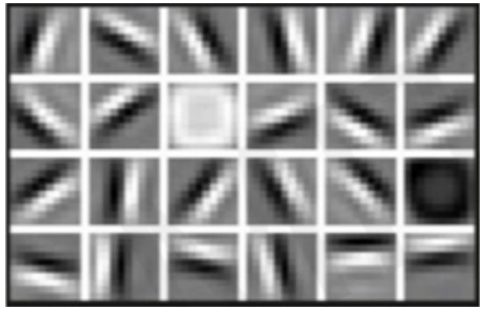
Why Deep Learning and Why now?

Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

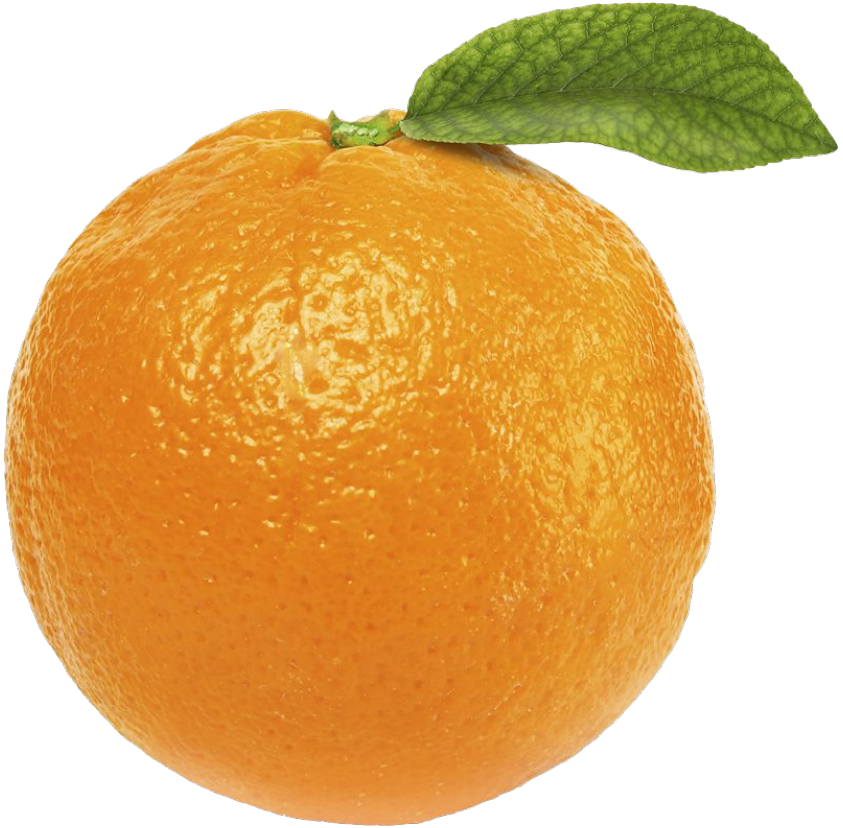
High Level Features

Facial Structure

Features

- Used to train an ML system
- Labels

Weight: 340g
Color: Orange



Why Now?

Neural Networks data back decades, so why the resurgence?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

1. Big Data
- Larger Datasets
- Easier Collection & Storage

2. Hardware
- GPUs
- Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

Data

Why is Data Important?

- Model-Centric vs Data-Centric
- Food for your network
- Model Performance Depends on Data

“What we call data are observations of real-world phenomena. […] Each piece of data provides a small window into a limited aspect of reality.”

- Page 1, Feature Engineering for Machine Learning, 2018.

Data Preparation

- Raw Data Must Be Prepared
- Extremely important (and tedious ;-;)

Main Concepts

- Handling Null Values
- Standardization
- Handling Categorical Variables
- One-Hot Encoding

Handling Null Values

- Life is never perfect
- Remove them or use imputation

```
df.dropna()
```

color	price
White	32.00
Blue	NaN
Blue	49.28
Red	10.23

color	price
White	32.00
Blue	49.28
Red	10.23

Imputation

- Process of substituting the missing values

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(df[['Weight']])
df['Weight'] = imputer.transform(df[['Weight']])
```

color	price
White	32.00
Blue	NaN
Blue	49.28
Red	10.23

color	price
White	32.00
Blue	30.50
Blue	49.28
Red	10.23

Normalization

- Transform values such that the values scale from 0 to 1

$$X' = \frac{X - X_{min}}{X_{\max} - X_{min}}$$

color	price
White	32.00
Blue	49.28
Red	10.23

color	price
White	0.55749
Blue	1
Red	0

Handling Categorical Variables

- Ordinal categorical variables
  - M < L < XL
- Nominal categorical variables
  - Blue < Green?

We need to preprocess ordinal and nominal categorical variables differently

Handling Ordinal Categorical Variables

Replace the categories with a corresponding integer

Can allow model to build relationships

```
size_mapping = {'M':1,'L':2}
df_cat['size'] = df_cat['size'].map(size_mapping)
```

Handling Nominal Categorical Variables

- Can’t do previous way (those relationships don’t exist)
- One-Hot Encoding

```
color_mapping = {0: “White”, 1: “Blue”, 2: “Red”}
```

color	price
White	32.00
Blue	49.28
Red	10.23

color	price
0	32.00
1	49.28
2	10.23

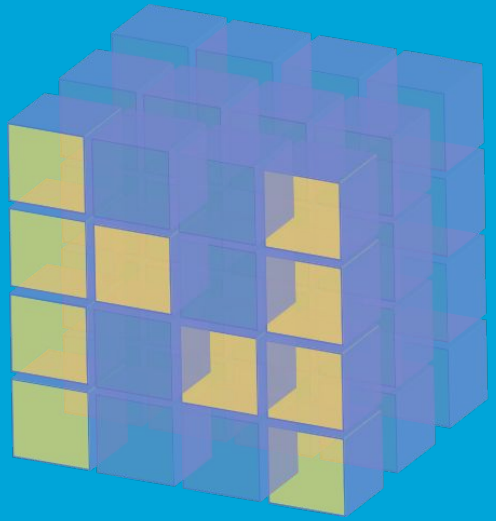
One-Hot Encoding

- Create *n* columns where *n* is the number of unique values that the nominal variable can take

$$X_{min} = 10.23 \qquad X_{max} = 49.28 \qquad X_{max} - X_{min} = 39.05$$

color	price
White	0.55749
Blue	1
Red	0

color_blue	color_white	color_red	price
0	1	0	0.55749
1	0	0	1
0	0	1	0



NumPy

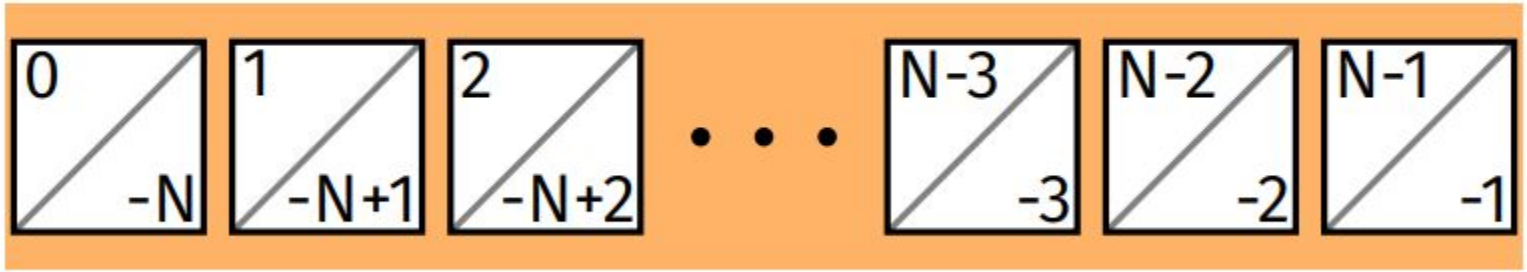
What is NumPy?

- Numerical Python
- Powerful array library

```
import numpy as np
```

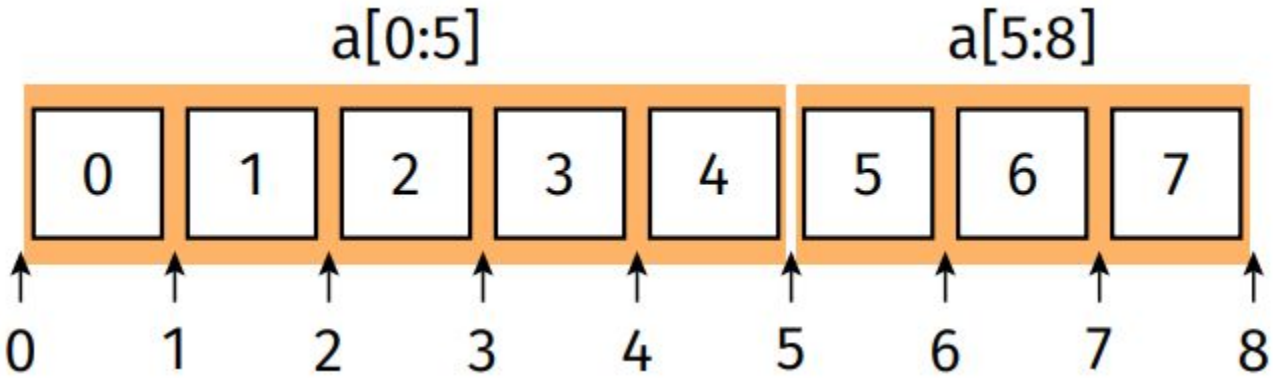
Crash Course!

- Indexing starts at 0
- Negative indices count from the end of the list to the beginning



List Slicing

- basic syntax: `[start:stop:step]`



- Step is defaulted to 1

Matrices

- List of Lists

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

```
matrix = [[0, 1, 2],
          [3, 4, 5],
          [6, 7, 8]]
```

- To extract item: matrix[row][column]
- To extract row: matrix[row]
- To extract column: ???

What now?

- List of Lists do not work like matrices
- Lists can contain arbitrary objects
- List elements can be scattered in memory

Introducing!

ais

ndarray

multidimensional, homogeneous array of fixed-size items

Indexing and Slicing - 1D

- Same as a normal Python list

Indexing and Slicing - Higher Dimensions

- Indexing has a comma separating dimensions

a[2, -3]

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

Indexing and Slicing - Higher Dimensions

a[:3, :5]

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

ais

a[:, 3]

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

a[1, 3:6]

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

a[1::2, ::3]

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

intro.nasuno.me

Member Attendance

1084

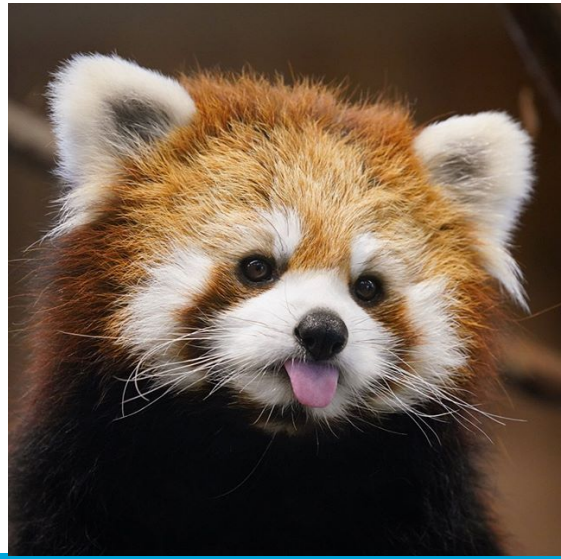
aisutd.org/member/attendance


pandas

What is it?

- One of the most widely used python libraries in data science
- Provides Dataframe object
- Like a spreadsheet with column names and row labels

```
import pandas as pd
```



Loading Data

- You will be doing this all the **time**

```
df = pd.read_csv("data/cereal.csv")
```

Viewing Data

- To view the top or bottom of the data, use head or tail

```
In [13]: df.head()
Out[13]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

In [14]: df.tail(3)
Out[14]:
                   A         B         C         D
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Viewing Data

- To view the the column titles, use columns (preferably)
- Index is used, but less commonly

```
In [15]: df.index
Out[15]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [16]: df.columns
Out[16]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

Selecting Data

- Much like a map in python, you select values by a key (which in pandas is the column label)

```
In [23]: df["A"]
Out[23]:
2013-01-01    0.469112
2013-01-02    1.212112
2013-01-03   -0.861849
2013-01-04    0.721555
2013-01-05   -0.424972
2013-01-06   -0.673690
Freq: D, Name: A, dtype: float64
```

intro.nasuno.me

Thank you for coming!