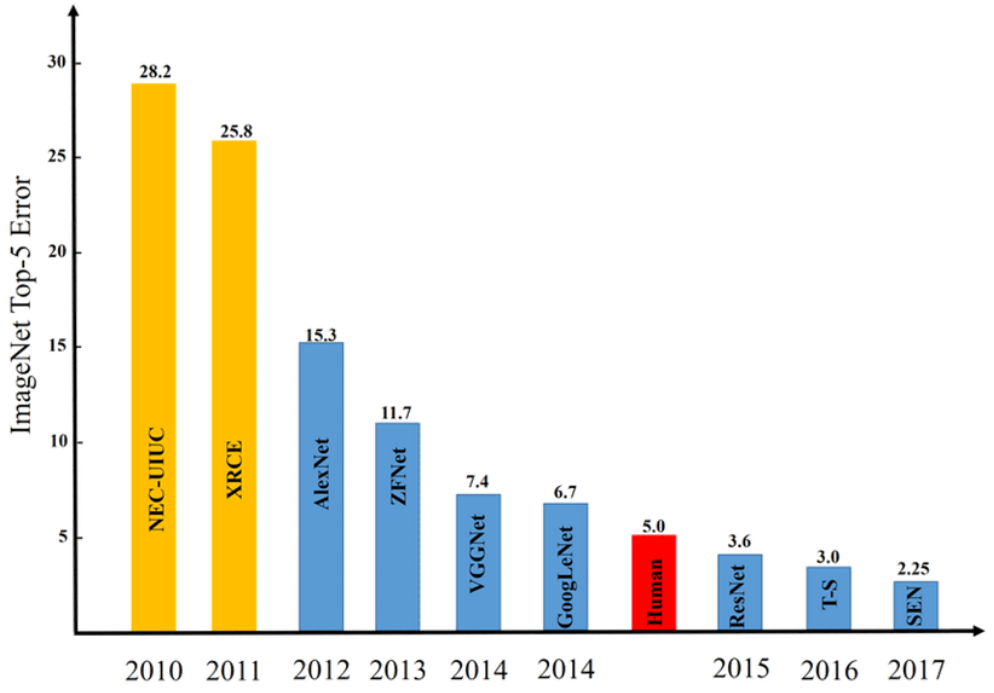
Deep Learning

Mohammad Reza Mohammadi

2021

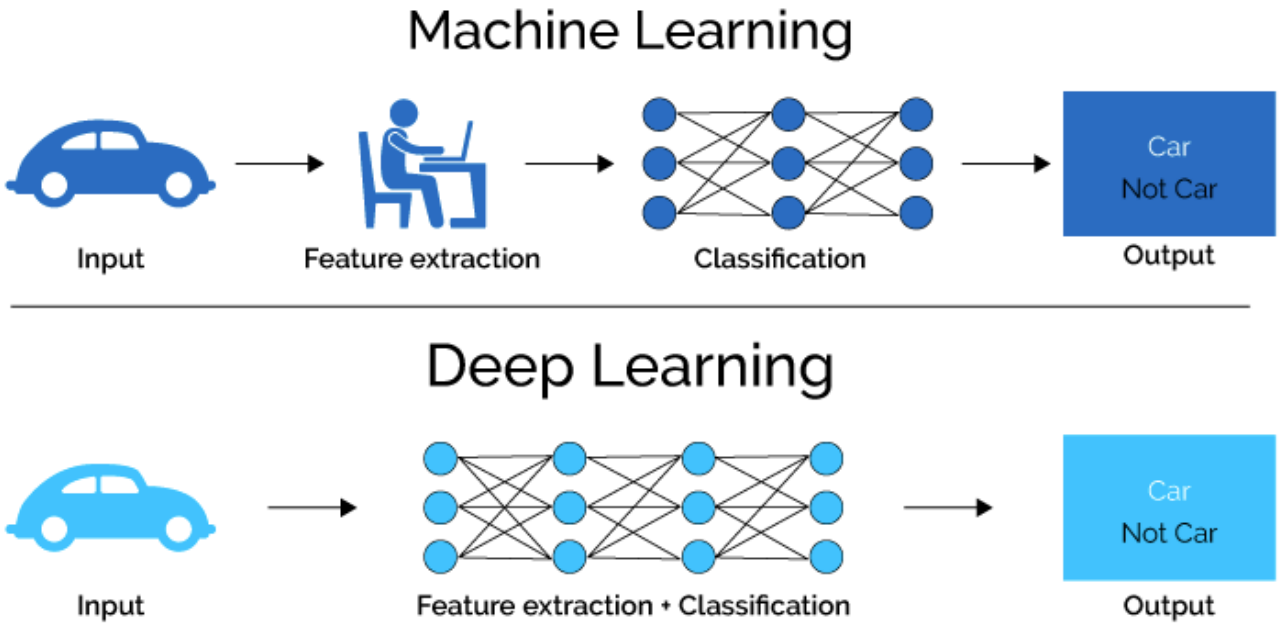
Return of Neural Networks

- ImageNet: a very difficult problem
  - classifying high resolution color images into 1,000 different categories after training on 1.4 million images
  - 2011: classical approaches, 25.8%
  - 2012: deep learning, 15.3% (huge breakthorough)
    - Since then, dominated by CNNs



What makes deep learning different?

- Easier, because it completely automates the feature engineering stage
- Earlier machine learning techniques required manual engineering of good layers of representations for their data
- Sophisticated multistage pipelines with a single, simple, end-to-end deep-learning model



https://lawtomated.com/a-i-technical-machine-vs-deep-learning/

Stacking shallow methods?

- Could shallow methods be applied repeatedly to emulate the effects of deep learning?
  - No, the deep model learns all layers of representation jointly
    - The optimal first representation layer in a three-layer model isn't the optimal first layer in a one-layer or two-layer model
- This is much more powerful than greedily stacking shallow models, because it allows for complex, abstract representations to be learned by breaking them down into long series of intermediate spaces (layers)

Why deep learning? Why now?

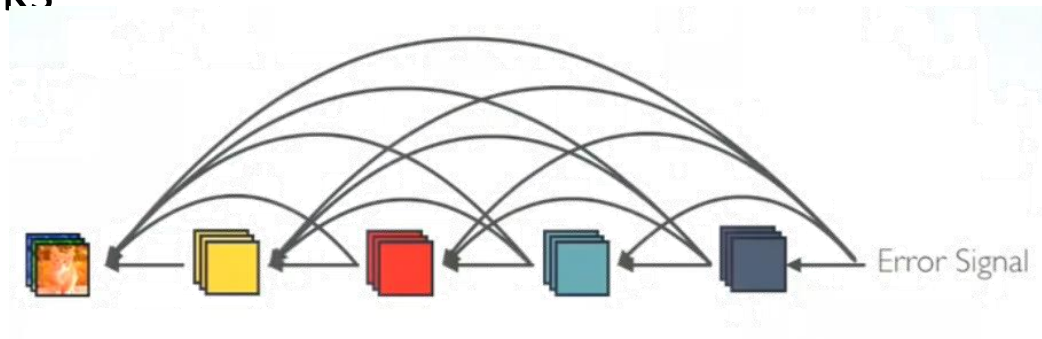
- Many of the algorithms are old:
  - CNN and backpropagation: 1989
  - LSTM: 1997

- What changed?
  - Hardware
  - Datasets and benchmarks
  - Algorithmic advances

flickr

YouTube

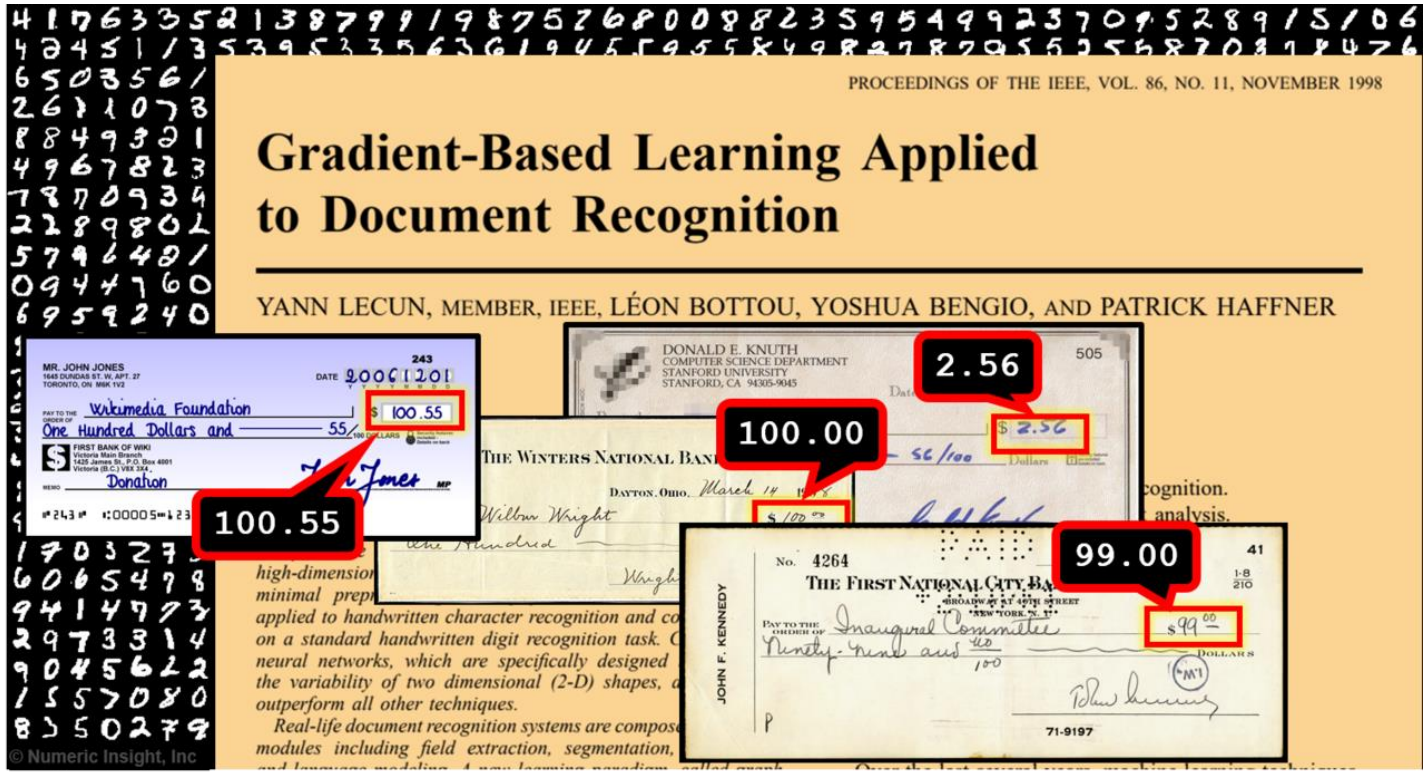
WIKIPEDIA
The Free Encyclopedia



Review: Neural Networks

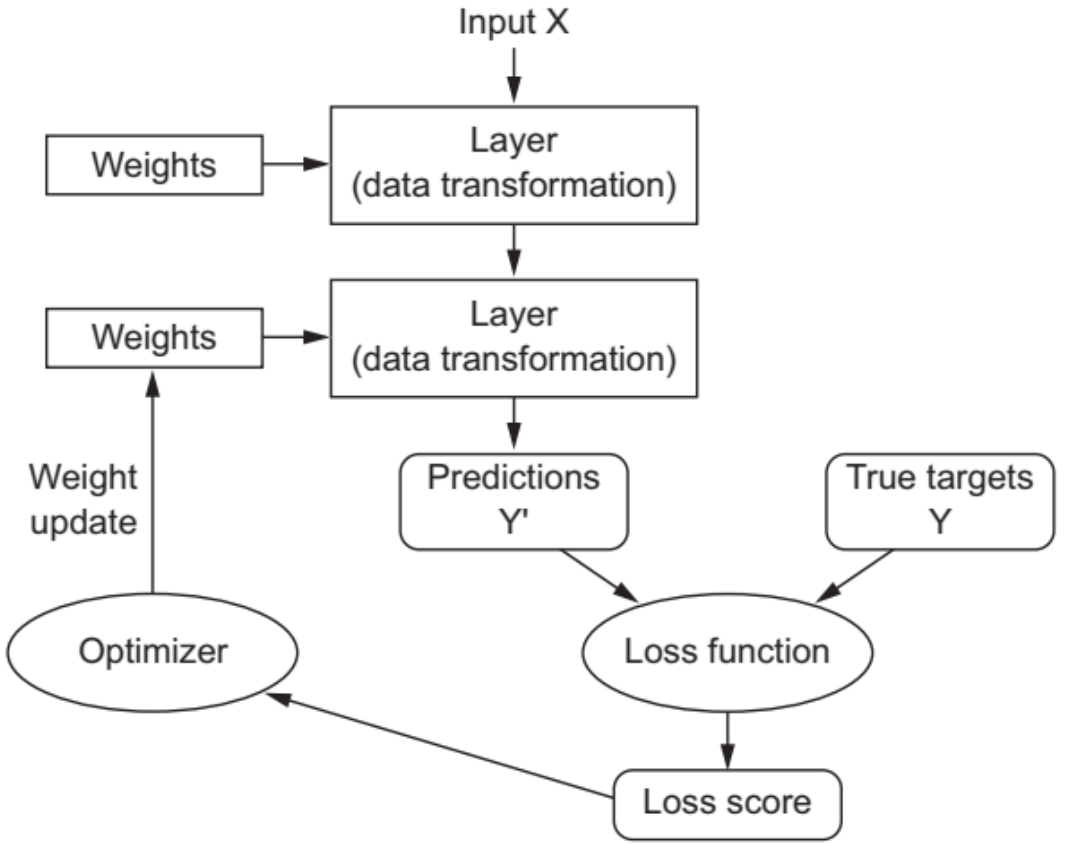
Our first network: handwritten recognition

- Digit classification
  - A 28x28 image into 10 categories (0 to 9)
- MNIST
  - An old classic dataset (1980s)
  - 60K training examples + 10K test

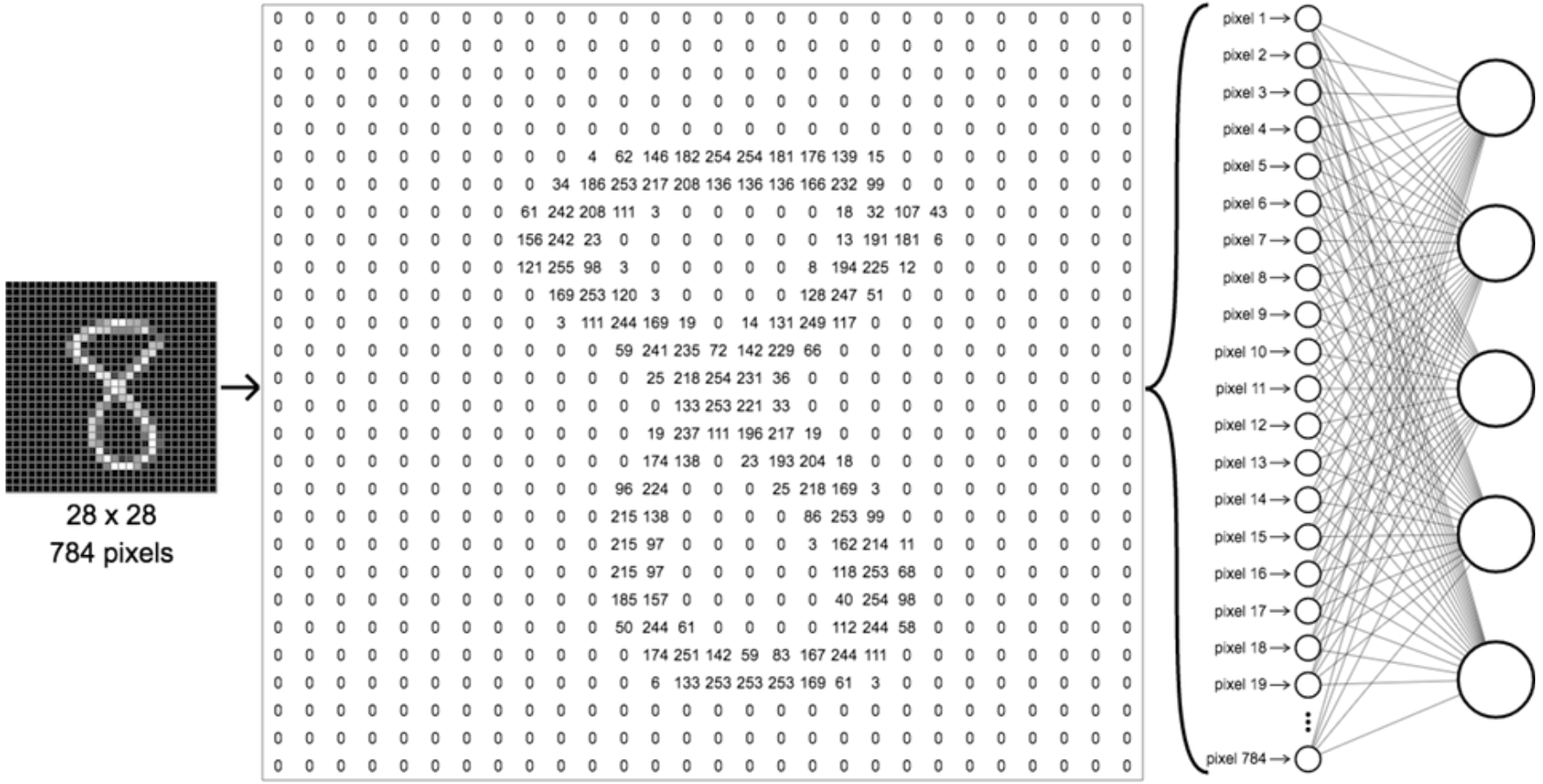


Our first network: handwritten recognition

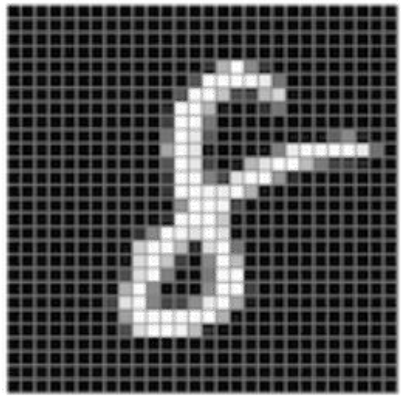
- The core building block of neural networks is the layer, a data-processing module that you can think of as a filter for data
- Layers extract representations out of the data fed into them that are more meaningful for the problem at hand
- Chaining together simple layers that will implement a form of progressive data distillation



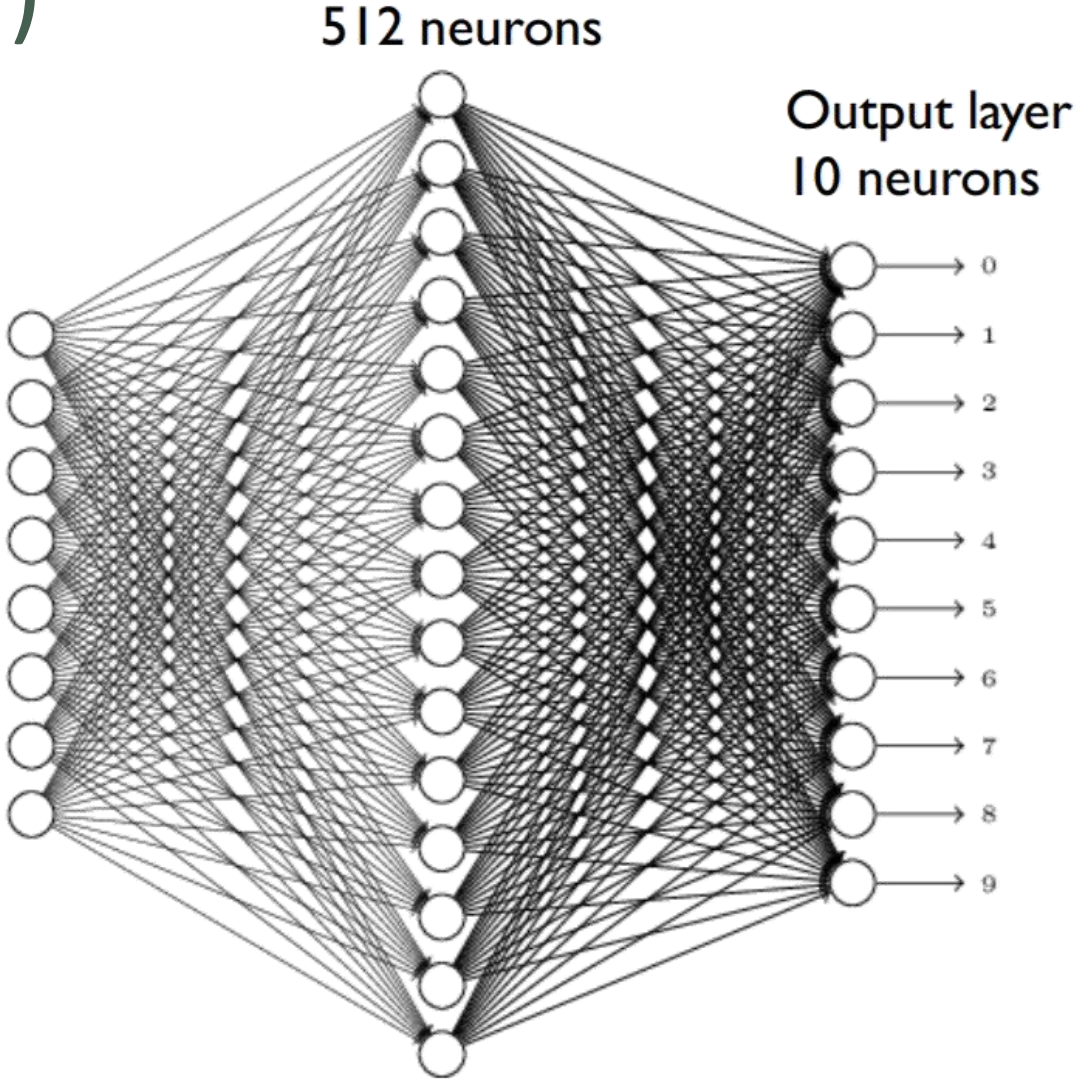
Fully Connected Layer



Multilayer perceptron (MLP)



Input layer
784 neurons



https://colab.research.google.com/drive/1fEklaDkh6_SrxrHP9FPOB93ndt3QngSt?usp=sharing

What are Tensors?

- Current machine-learning systems use **tensors** as their basic data structure
  - A container for data - almost always numerical data
  - Matrices are 2D tensors - Tensors are a generalization of matrices to an arbitrary number of dimensions
  - In the context of tensors, a dimension is often called an axis

Scalars and vectors

- A tensor that contains only one number is called a scalar (or scalar tensor, or 0-dimensional tensor, or 0D tensor)

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

- An array of numbers is called a vector, or 1D tensor

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

Matrices

- An array of vectors is a matrix, or 2D tensor

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

- The entries from the first axis are called the rows, and the entries from the second axis are called the columns

3D tensors

- If you pack such matrices in a new array, you obtain a 3D tensor

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

Manipulating tensors in Numpy

- Selecting specific elements in a tensor is called tensor slicing
  - Select digits #10 to #100 (#100 isn't included) and put them in an array of shape (90, 28, 28):

```
>>> my_slice = train_images[10:100]
>>> print(my_slice.shape)
(90, 28, 28)
```

  - Equivalently (: is equivalent to selecting the entire axis)

```
>>> my_slice = train_images[10:100, :, :]
>>> my_slice.shape
(90, 28, 28)
>>> my_slice = train_images[10:100, 0:28, 0:28]
>>> my_slice.shape
(90, 28, 28)
```

Manipulating tensors in Numpy

- Select 14x14 pixels in the bottom-right corner of all images

```
my_slice = train_images[:, 14:, 14:]
```

- Crop the images to patches of 14x14 pixels centered in the middle
  - Much like negative indices in Python lists, they indicate a position relative to the end of the current axis

```
my_slice = train_images[:, 7:-7, 7:-7]
```

Data batch

- Deep-learning models don't process an entire dataset at once; rather, they break the data into small batches
  - Concretely, here's one batch of our MNIST digits, with batch size of 128

```
batch = train_images[:128]
```

  - And the next batch:

```
batch = train_images[128:256]
```

  - And the n^{th} batch:

```
batch = train_images[128 * n:128 * (n + 1)]
```