Neural Networks for Data Science Applications

Master's Degree in Data Science

Lecture 0: Introduction to the course

Lecturer: S. Scardapane

SAPIENZA
UNIVERSITÀ DI ROMA

(Deep) neural networks are **composable**, **differentiable** functions that can be **optimized end-to-end** numerically.

Introduction

A simple example

A freshly-graduated data scientist is assigned two tasks:

Task 1

Please filter these emails according to the size of their attachments.

Task 2

Please filter these clients according to how much we should trust them.

Task 1 can be coded classically, while task 2 probably requires some **machine learning**.

Trust is not easy to define, but it is trivial with the benefit of hindsight. For example, if clients default on a loan, they were (probably) *untrustworthy*.

Machine learning (ML) algorithms exploit this by *inferring* the relation from *historical* data. A variety of techniques exist: kernel methods, boosted trees, k-NN, rule-based models...

However, not all data is the same, and most methods are not suitable when dealing with high-dimensional objects.

Case 1

A client is described by date of birth, age, nationality, and money over the last six months.

Cool! I'll just use my favourite random forest implementation.

```
from sklearn.ensemble import RandomForestClassifier
my_classifier = RandomForestClassifier().fit(X, y)
# Done!
```

Case 2

Like case 1, but we are buying a 10000-dimensional descriptor of the client from some external, undocumented third party.

Maybe some sparse, high-dimensional classifier?

```
#from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Lasso
my_classifier = Lasso().fit(X, y)
# Does it even fit in memory?
```

Case 3

Like case 2, but we are adding their social feed, filming their interviews, and we have stolen their clinical folders.

Wait, what?

```
#from sklearn.ensemble import RandomForestClassifier
#from sklearn.linear_model import Lasso#
???
```

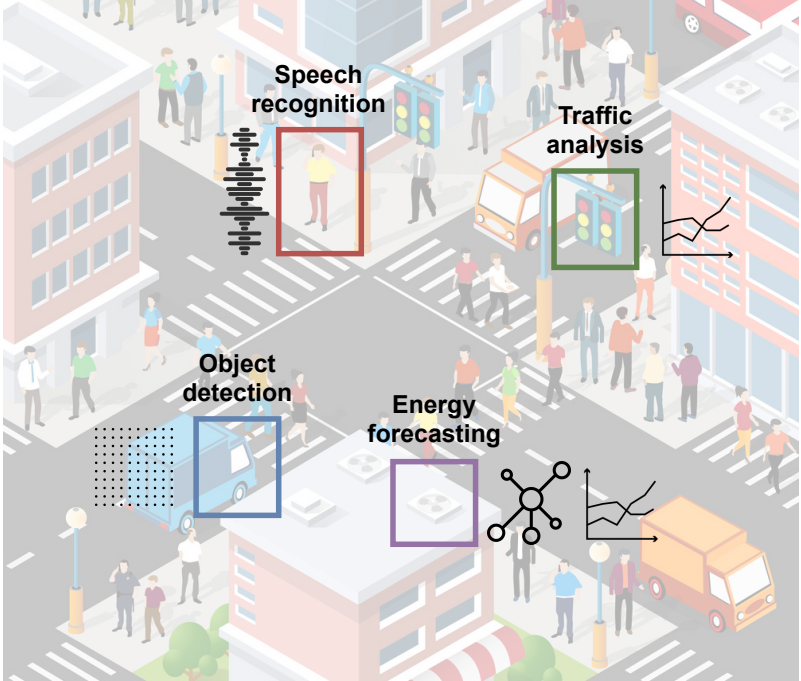
In order to use ML in case 3, we need techniques that:

1. can properly model **raw, high-dimensional data** such as images, videos, text, graphs, ...;
2. can **combine multiple components** in a modular fashion;
3. can be trained efficiently from **large amounts of data**.

Neural networks are, to this day, the only algorithms that satisfy points (1)-(3) simultaneously.

Introduction

Neural networks are (almost) everywhere


Speech recognition
Traffic analysis
Object detection
Energy forecasting

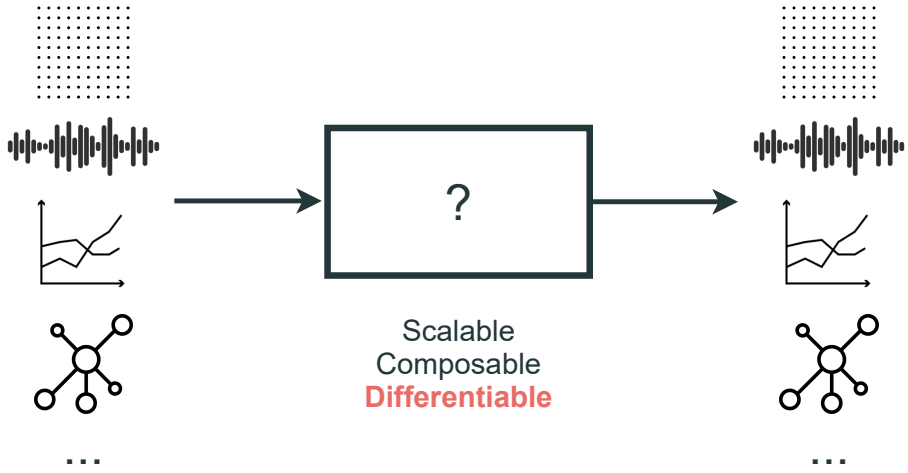
Listing all notable applications of neural networks is almost impossible: think of a complex problem, and someone has probably developed a state-of-the-art model for it, ranging from **neural translation** to **protein folding**, **videogame playing**, **neural rendering**, **physics simulations**, ...

Amazingly, all this is powered by a very small set of components and organizing principles (e.g., differentiability, invariances and equivariances, sparsity, locality). **Data**, **computing**, and **software** are keys.

Hint: browse `https://paperswithcode.com/sota` for a few examples.

Introduction

Defining a neural network


?
Scalable
Composable
Differentiable

Imagine there is an error in a Python program: a certain output does not correspond to your **desired output**. The only way to solve this is manual inspection and debugging.

However, for a **differentiable** program f, we can always use **numerical optimization** to solve $\max L(f(x))$ or $\min L(f(x))$, where L encodes what we would like the program to do (typically based on a set of examples).

Working with differentiable functions is limiting: we can manipulate real numbers of **collections** or real numbers (tensors, sets, sequences, graphs), and we are also restricted to a small number of operations (e.g., $x < \tau$ is not differentiable).

In a broad sense, **deep learning** (aka **differentiable programming**) studies how far we can get under this constraint.

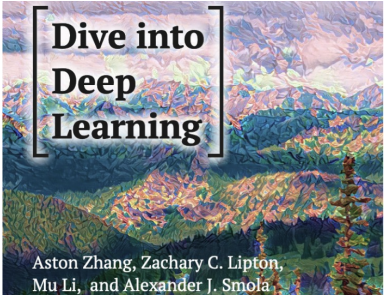
About the course

Organization and material

- Understand the required math to easily manipulate deep learning models and algorithms;
- Learn and apply some fundamental components (convolutions, attention, ...);
- Become proficient in a real-world deep learning library (**TensorFlow**);
- Being conscious of some serious limitations (bias, brittleness), and current research lines (self-supervision, MLOps, ...).

1. **Preliminaries** (tensors, linear algebra, optimization).
2. **Supervised learning** as numerical optimization (stochastic gradient descent);
3. Linear models and **fully-connected** models.
4. **Convolutional models** (for sequential, spatial, and temporal data).
5. **Tricks** to train deeper models (dropout, batch normalization, ...).
6. **Attention** models (and **recurrent** models, *optional*).
7. **Graph** models (e.g., graph convolutional networks).
8. Mitigating **bias** and **adversarial attacks**.
9. *(If we have time)* **Continual** and **self-supervised** learning (hints).

1. Several practical lectures with **TensorFlow** (hands-on coding from scratch).
2. When possible, a showcase of other libraries (e.g., HuggingFace Datasets).
3. Learning to containerize and deploy models (hints).



The book is **free to read** on the web, and can be downloaded as a set of **executable notebooks**. Each slide will mention the corresponding sections.

Official course website:
`https://www.sscardapane.it/teaching/nnds-2022/`.

Please register to the **Google Classroom** from the website for all updates.

- One **mid-term** homework (5 points).
- One **final project** (10-15 points, depending on whether the mid-term homework was delivered).
- One **oral examination** on the program (15 points).

If time allows, a few (optional) small exercises from Google Classroom.