EDAN96
Applied Machine Learning
Lecture 8: Classification Techniques and Neural Networks

Pierre Nugues

Pierre.Nugues@cs.lth.se

November 27, 2024

Content

Overview and practice of some neural network architectures:

- Datasets
- Input formatting
- Feedforward networks

We will use:

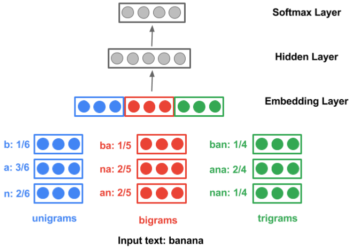
- PyTorch, `https://pytorch.org`, a powerful API to design and train network, and
- scikit-learn, `https://scikit-learn.org/stable/`, a general purpose machine-learning toolkit.

Goal of the Lecture

Describe a language detector: Given a string predict the language:

- *Bonjour* –> French
- *Guten Tag* –> German

Follow the complete compact language detector (CLD3)
`https://github.com/google/cld3`



You will implement it during the fourth lab

Dataset Exploration

When dealing with a dataset, the first step is to explore it.
All machine-learning experiments should start with it.
Such a step is called *exploratory data analysis* (EDA)
It consists of measuring:

- the size of the dataset (observations)
- the number of classes, number of observations per class
- the mean and standard deviation of the variables
- the domain of categorical classes
- number of missing values, etc.

EDA and Visualization

Many EDAs show results on graphics

Many examples on `https://www.kaggle.com/`, for instance `https://www.kaggle.com/code/imoore/intro-to-exploratory-data-analysis-eda-in-python` and `https://www.kaggle.com/code/ekami66/detailed-exploratory-data-analysis-with-python`

The Tatoeba Dataset

- A language detector (lab 4) is a kind of categorization
- Categorization is one of the most popular applications of machine learning
- See here for the datasets: `https://huggingface.co/tasks`, `https://archive.ics.uci.edu/ml/datasets.php`
- As dataset to train CLD3, we will use Tatoeba `https://tatoeba.org/`

Code Example

Experiment: An elementary EDA Jupyter Notebook:
`https://github.com/pnugues/edan96/blob/main/programs/5-tatoeba_eda_select.ipynb`
up to *Splitting the corpus*

Dataset Formatting

When training a model, a common practice is to split the dataset in three sets:

1. Train, validation (also called development), and test sets
2. Already done in many datasets, see `https://huggingface.co/datasets`
3. Otherwise do it. You will have to do it for CLD3

This partitioning is essential to measure overfit when training.
Identical test sets give means to compare results and performance of systems (see scientific papers).

Code Example

Experiment: An elementary EDA Jupyter Notebook:
https://github.com/pnugues/edan96/blob/main/programs/5-tatoeba_eda_select.ipynb

Input Formatting

Starting from observations, we format the input to use it as input to a classifier

This step is also called the vectorization: Turn the input into numerical vectors

Numerical input: We already have vectors or tensors to represent our data. We have to standardize them

Categorical or text input: Linear classifiers do not accept this: We have to encode the symbols as vectors. Using unit vectors is a baseline technique. It is called one-hot encoding.

Hashing: One-hot encoding may result in very large vectors, millions of parameters, when dealing with text. We can reduce their size with hashing techniques

Embeddings: One-hot encoding is a sparse representation. We can design network architectures to produce smaller (dim=100) and dense vectors

Standardization

Most algorithms in classification do not work well if the parameters have widely differing ranges:

1. Parameter 1: $[10,000,000; 20,000,000]$
2. Parameter 2: $[-0.01; 0.002]$

Before training a model, we must standardize the data.
Two options:

1. Remove the mean and divide by the standard deviation with respect to columns:
$$x_{i,j_{std}} = \frac{x_{i,j} - \bar{x}_{.,j}}{\sigma_{x_{.,j}}}.$$
2. Norm the rows to 1:
$$x_{i,j_{norm}} = \frac{x_{i,j}}{\sqrt{\sum_{k=0}^{n-1} x_{i,k}^2}}.$$

Fisher's Iris Dataset (1936)

180 MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS

Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8
4·9	3·1	1·5	0·1	5·2	2·7	3·9	1·4	7·2	3·6	6·1	2·5
5·4	3·7	1·5	0·2	5·0	2·0	3·5	1·0	6·5	3·2	5·1	2·0
4·8	3·4	1·6	0·2	5·9	3·0	4·2	1·5	6·4	2·7	5·3	1·9
4·8	3·0	1·4	0·1	6·0	2·2	4·0	1·0	6·8	3·0	5·5	2·1
4·3	3·0	1·1	0·1	6·1	2·9	4·7	1·4	5·7	2·5	5·0	2·0
5·8	4·0	1·2	0·2	5·6	2·9	3·6	1·3	5·8	2·8	5·1	2·4
5·7	4·4	1·5	0·4	6·7	3·1	4·4	1·4	6·4	3·2	5·3	2·3
5·4	3·9	1·3	0·4	5·6	3·0	4·5	1·5	6·5	3·0	5·5	1·8
5·1	3·5	1·4	0·3	5·8	2·7	4·1	1·0	7·7	3·8	6·7	2·2
5·7	3·8	1·7	0·3	6·2	2·2	4·5	1·5	7·7	2·6	6·9	2·3
5·1	3·8	1·5	0·3	5·6	2·5	3·9	1·1	6·0	2·2	5·0	1·5
5·4	3·4	1·7	0·2	5·9	3·2	4·8	1·8	6·9	3·2	5·7	2·3
5·1	3·7	1·5	0·4	6·1	2·8	4·0	1·3	5·6	2·8	4·9	2·0

Code Example

Experiment: sklearn and its cancer dataset with a Jupyter Notebook:
`https://github.com/pnugues/edan96/blob/main/programs/6-standardization.ipynb`

Categorical Values

Linear classifiers only understand numbers

A collection of two documents D1 and D2:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

How to represent these words or these documents?

Categorical Values: One-hot encoding

Let us suppose we want to predict the parts of speech of the words, for instance *plans* in:

Input:	Chrysler	*plans*	major	investments	in	Mexico
Output:	PNoun	Verb	Adjective	Noun	Prep.	PNoun

One-hot encoding:

1. Assigns a unique index to each symbol. The number of indices corresponds to the number of symbols:
   `{'Chrysler': 1, 'in': 2, 'investments': 3, 'major': 4, 'Mexico': 5, 'plans': 6}`
2. Represent a symbol of index i by a unit vector $\mathbf{x} = (x_1, x_2, ..., x_n)$, where n is the largest index and all the coordinates are 0, except $x_i = 1$

```
'Chrysler':    (1, 0, 0, 0, 0, 0)
'in':          (0, 1, 0, 0, 0, 0)
'investments': (0, 0, 1, 0, 0, 0)
...
```

Categorical Values: Multi-hot encoding

A collection of two documents D1 and D2:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

Multi-hot encoding (also called a bag-of-words representation):

1. Creates a index of all the symbols (words) in all the documents
2. For each document, creates a set of its symbols (word)
3. Represents a document by a vector of 0s and 1s. $x_i = 1$ if the word of index i is in the document, or 0 otherwise.

D.	america	chrysler	in	investments	latin	major	mexico	new	plans
1	1	1	1	1	1	0	0	1	1
2	0	1	1	1	0	1	1	0	1

Hashing

One-hot encoding leads to very large dimensions as there are billions of different words.

In the Tatoeba experiment (see notebook), the number of unigrams, bigrams, and trigrams is (4843, 88601, 299572)

A solution is to hash the symbols and use the remainder of a division (modulo) as index

```
>>> hash('abc')
-6712881850779232724
>>> hash('abc') % 100
76   # Always less than 100
```

Hashing:

1. Reduces the vectors size and makes it manageable
2. Creates conflicts: two symbols can have the same hash numbers
3. Is usable in classification

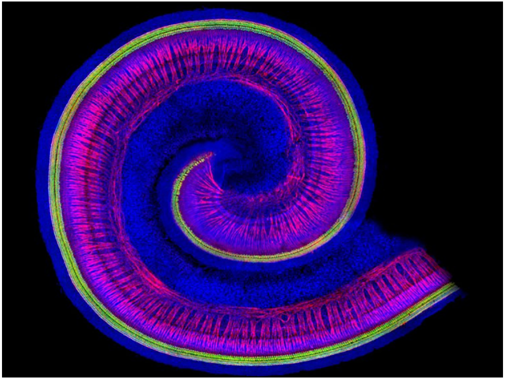
Code Example

Experiment: hashing CLD3 n-grams Jupyter Notebook:
https://github.com/pnugues/edan96/blob/main/programs/7-ngram_hashing.ipynb

Reproducible Hash Codes

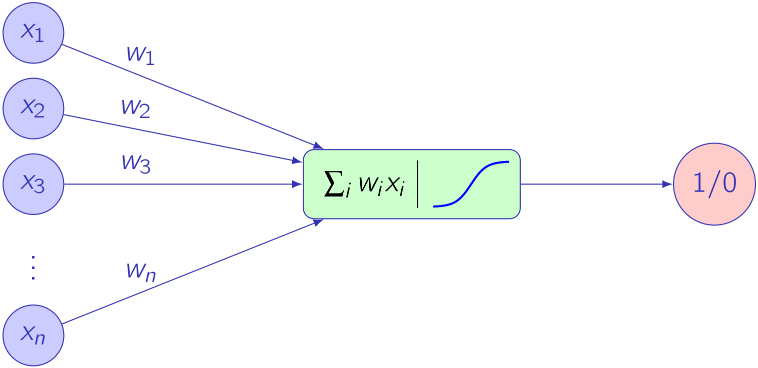
```
def reproducible_hash(string):
    """
    reproducible hash on any string

    Arguments:
       string: python string object

    Returns:
       signed int64
    """

    # We are using MD5 for speed not security.
    h = hashlib.md5(string.encode("utf-8"),
                    usedforsecurity=False)
    return int.from_bytes(h.digest()[0:8], 'big', signed=True)
```

Neural Networks



A photomicrograph showing the classic view of the snail-shaped cochlea with hair cells stained green and neurons showing reddish-purple. [Decibel Therapeutics (`https://www.decibeltx.com`)]. Source: `https://www.genengnews.com/insights/targeting-the-inner-ear/`

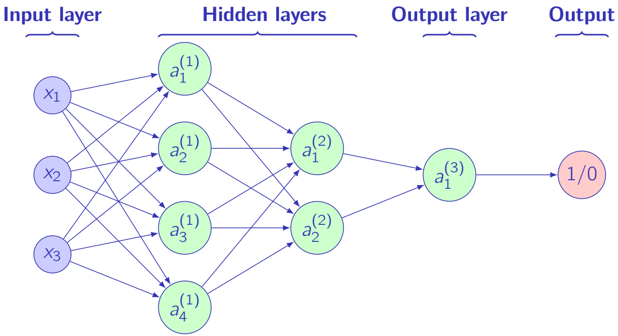
Logistic Regression as a Neural Network



$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x})$$

The complete linear model also includes a bias.

Neural Networks



$$
\begin{aligned}
\hat{y} &= W_3 W_2 W_1 \mathbf{x} \\
&= \mathbf{x} W_1^\intercal W_2^\intercal W_3^\intercal
\end{aligned}
$$

The second equation is more natural as it extends to a batch:

$$\hat{y} = XW_1^\intercal W_2^\intercal W_3^\intercal$$

See `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html`

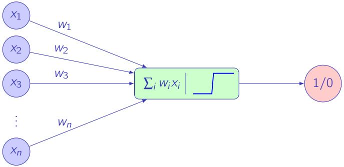
Activation Functions

This simple architecture is equivalent one single matrix.

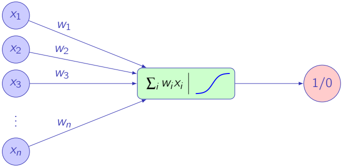
To build a network, we add activation functions in the nodes, that is after each matrix application.

Usual activation functions:

- The perceptron



- Logistic regression

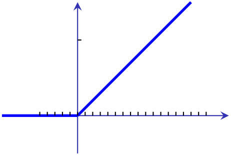


We normally apply logistic regression in the last layer.

Activation Functions

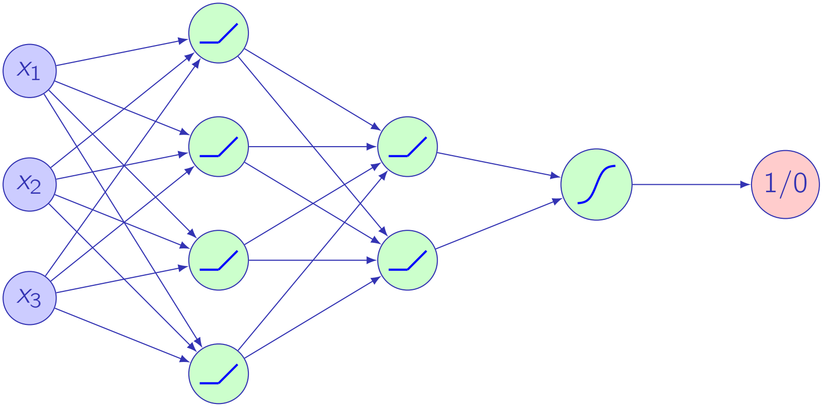
The other layers typically use a rectified linear unit (ReLU) or a variant of it

$$\mathrm{ReLU}(x) = \max(0, x).$$



This creates nonlinearities in the model and enables it to map the input space into the class space.

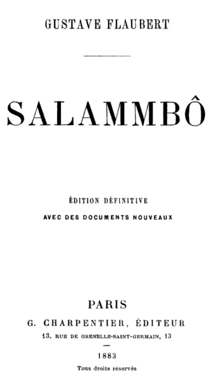
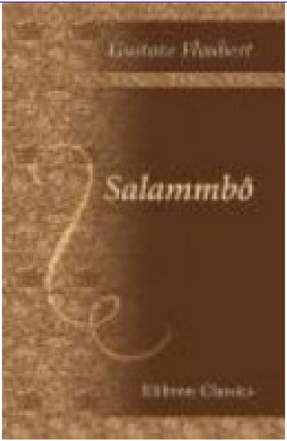
Neural Networks with Hidden Layers



$$\begin{aligned}\hat{y} &= \sigma(W_3\,\mathrm{ReLU}(W_2\,\mathrm{ReLU}(W_1\mathbf{x})))\\ &= \sigma(\mathrm{ReLU}(\mathrm{ReLU}(\mathbf{x}W_1^\intercal)W_2^\intercal)W_3^\intercal)\end{aligned}$$

A Text Dataset: *Salammbô*

A corpus is a collection – a body – of texts.

French original	English translation
	

GUSTAVE FLAUBERT

SALAMMBÔ

ÉDITION DÉFINITIVE
AVEC DES DOCUMENTS NOUVEAUX

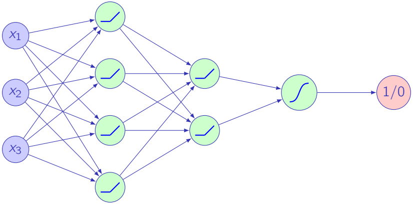
PARIS
G. CHARPENTIER, ÉDITEUR
13, RUE DE GRENELLE-SAINT-GERMAIN, 13
—
1883
Tous droits réservés

Classification Dataset

Dataset for binary classification: *Salammbô* in French (1) and English (0)

	# char.	# A	class (y)	# char.	# A	class (y)
Chapter 1	36,961	2,503	1	35,680	2,217	0
Chapter 2	43,621	2,992	1	42,514	2,761	0
Chapter 3	15,694	1,042	1	15,162	990	0
Chapter 4	36,231	2,487	1	35,298	2,274	0
Chapter 5	29,945	2,014	1	29,800	1,865	0
Chapter 6	40,588	2,805	1	40,255	2,606	0
Chapter 7	75,255	5,062	1	74,532	4,805	0
Chapter 8	37,709	2,643	1	37,464	2,396	0
Chapter 9	30,899	2,126	1	31,030	1,993	0
Chapter 10	25,486	1,784	1	24,843	1,627	0
Chapter 11	37,497	2,641	1	36,172	2,375	0
Chapter 12	40,398	2,766	1	39,552	2,560	0
Chapter 13	74,105	5,047	1	72,545	4,597	0
Chapter 14	76,725	5,312	1	75,352	4,871	0
Chapter 15	18,317	1,215	1	18,031	1,119	0

Neural Networks with Hidden Layers in PyTorch




```
model = nn.Sequential(
    nn.Linear(input_dim, 4),
    nn.ReLU(),
    nn.Linear(4, 2),
    nn.ReLU()
    nn.Linear(2, 1),
    torch.nn.Sigmoid()
    )
```

Code Example

We extend our logistic regression program to a neural network

Experiment: Jupyter Notebook: https://github.com/pnugues/edan96/blob/main/programs/8-Salammbo_class_torch.ipynb