Introduction to DL and OCR

Day 4

Morning: Sophie Robert-Hayek
Afternoon: Guillaume Charbonnier

University of Lorraine | Araymond

Py4SHS 2023

Reminders of previous session

Outline

Reminder on supervised learning

Question

Can you remind me of what is **supervised learning** ?

Reminder on supervised learning

Question

Can you remind me of what is **supervised learning** ?

Question

Can you remind me of possible applications of **supervised learning** to digital humanities ?

Reminder on supervised learning

Question

Can you remind me of what is **supervised learning** ?

Question

Can you remind me of possible applications of **supervised learning** to digital humanities ?

Today, we will be learning about *Deep Learning* (DL) and its application to *Optical Character Recognition* (OCR).

Optical Characteristics Recognition

Outline

Introduction to OCR

Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text.

Introduction to OCR

Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text.

OCR enables efficient digitization and data extraction from printed or handwritten text, for example from:

- a scanned document
- a scene photo (text on signs and billboards in a landscape photo ...)
- subtitle text superimposed on an image (subtitles on movie ...).

Early Concepts of Character Recognition

- Early attempts at character recognition date back **as early as 1914**.
- Early systems used **mechanical devices** to recognize characters on telegraph signals and punch cards.
- These systems were limited to specific fonts and often struggled with variations in handwriting and typefaces.

Emergence of Electronic OCR

- The development of electronic computers in the mid-20th century paved the way for more advanced OCR techniques.
- In the 1950s and 1960s, researchers began experimenting **with pattern recognition and template matching algorithms**: mostly for recognizing **machine-printed characters in controlled environments** (price tags, passports ...).
- The 1970s and 1980s saw significant progress in OCR technology, as Machine Learning models improved.

Optacon

1954-1964: The Optacon, the first portable OCR device

Modern OCR and Machine Learning

The late 20th century and early 21st century brought about a **revolution in OCR because of machine learning**:

Modern OCR and Machine Learning

The late 20th century and early 21st century brought about a **revolution in OCR because of machine learning**:

- Machine and Deep learning algorithms became more and more efficient.

Modern OCR and Machine Learning

The late 20th century and early 21st century brought about a **revolution in OCR because of machine learning**:

- Machine and Deep learning algorithms became more and more efficient.
- Data collection, storage and labeling exploded.

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

- Document digitization and archiving

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

- Document digitization and archiving
- Self-driving cars

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

- Document digitization and archiving
- Self-driving cars
- Text extraction for search engines

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

- Document digitization and archiving
- Self-driving cars
- Text extraction for search engines
- Mobile apps for scanning and translating text

Current and Future Applications

Today, OCR is integrated into various applications and is present in our day to day life, including:

- Document digitization and archiving
- Self-driving cars
- Text extraction for search engines
- Mobile apps for scanning and translating text
- Automated data entry using mobile camera

Challenges and Limitations

- Despite significant advancements, OCR still faces challenges:
  - Accurate recognition of handwriting with diverse styles
  - Complex layouts and formatting in documents
  - Degraded or poor-quality source material

Challenges and Limitations

- Despite significant advancements, OCR still faces challenges:
  - Accurate recognition of handwriting with diverse styles
  - Complex layouts and formatting in documents
  - Degraded or poor-quality source material
- Ongoing research focuses on:
  - **Improving accuracy**;
  - **Handling more languages**;
  - **Enhancing handwritten text recognition**;
  - **Combining OCR with other technologies such as natural language processing and image preprocessing**.

Challenges and Limitations

- Despite significant advancements, OCR still faces challenges:
  - Accurate recognition of handwriting with diverse styles
  - Complex layouts and formatting in documents
  - Degraded or poor-quality source material
- Ongoing research focuses on:
  - **Improving accuracy**;
  - **Handling more languages**;
  - **Enhancing handwritten text recognition**;
  - **Combining OCR with other technologies such as natural language processing and image preprocessing**.

OCR is not a **solved** problem, especially when it comes to handwritten data.

OCR and digital humanities

Applying OCR to manuscript data presents unprecedented opportunities:

OCR and digital humanities

Applying OCR to manuscript data presents unprecedented opportunities:

- **Digital Access:** Enabling scholars to search, analyze, and study manuscripts remotely.

OCR and digital humanities

Applying OCR to manuscript data presents unprecedented opportunities:

- **Digital Access:** Enabling scholars to search, analyze, and study manuscripts remotely.
- **Preservation**: Reducing the need for physical handling of delicate manuscripts.

OCR and digital humanities

Applying OCR to manuscript data presents unprecedented opportunities:

- **Digital Access:** Enabling scholars to search, analyze, and study manuscripts remotely.
- **Preservation**: Reducing the need for physical handling of delicate manuscripts.
- **Text Mining**: Extracting insights from large collections for research purposes.

Challenges in OCR for Ancient Manuscripts/Prints

- **Complex Scripts**: Variability in handwriting styles, ligatures, illustrations, and diacritics.

Challenges in OCR for Ancient Manuscripts/Prints

- **Complex Scripts**: Variability in handwriting styles, ligatures, illustrations, and diacritics.
- **Abbreviations**: Widespread use of abbreviation marks and conventions.

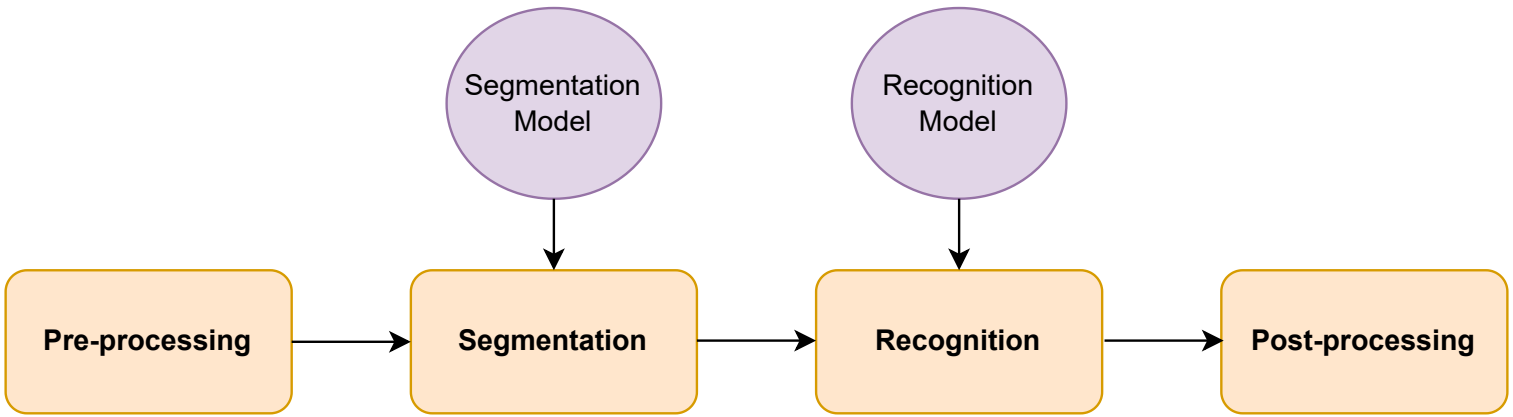
Challenges in OCR for Ancient Manuscripts/Prints

- **Complex Scripts**: Variability in handwriting styles, ligatures, illustrations, and diacritics.
- **Abbreviations**: Widespread use of abbreviation marks and conventions.
- **Degraded Materials**: Manuscripts can be damaged, faded, or deteriorated.

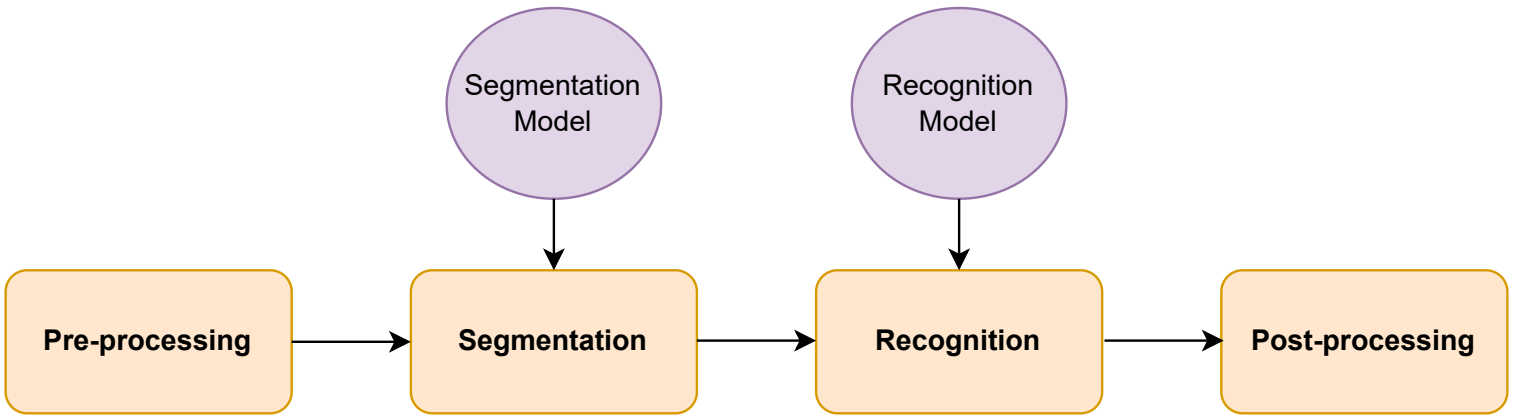
Challenges in OCR for Ancient Manuscripts/Prints

- **Complex Scripts**: Variability in handwriting styles, ligatures, illustrations, and diacritics.
- **Abbreviations**: Widespread use of abbreviation marks and conventions.
- **Degraded Materials**: Manuscripts can be damaged, faded, or deteriorated.
- **Multilingual Text**: Manuscripts might contain multiple languages or dialects.

OCR pipeline



OCR pipeline



Basing yourselves on yesterday lecture, what kind of learning models are the Segmentation model and Recognition model ?

Major steps of OCR

1. **Image Preprocessing:** Preprocess to improve the readability of the image.

Major steps of OCR

1. **Image Preprocessing:** Preprocess to improve the readability of the image.
2. **If using "classical" approach**:
   - **Character Segmentation:** segments the characters.
   - **Character Recognition:** recognize characters.

Major steps of OCR

1. **Image Preprocessing:** Preprocess to improve the readability of the image.
2. **If using "classical" approach**:
   - **Character Segmentation:** segments the characters.
   - **Character Recognition:** recognize characters.
3. **If using "newer" approach**:
   - **Text Line Recognition:** segment the text into individual lines.
   - **Perform word recognition:** match the line into a set of words.

Major steps of OCR

1. **Image Preprocessing:** Preprocess to improve the readability of the image.
2. **If using "classical" approach**:
   - **Character Segmentation:** segments the characters.
   - **Character Recognition:** recognize characters.
3. **If using "newer" approach**:
   - **Text Line Recognition:** segment the text into individual lines.
   - **Perform word recognition:** match the line into a set of words.
4. **Postprocessing:** check the output using language models.

Preprocessing

- **Binarization**: convert grayscale image into a binary image, with each pixel classified as foreground (usually black) or background (usually white), to **create a clear distinction between the text elements and the paper or background** on which the text is written.

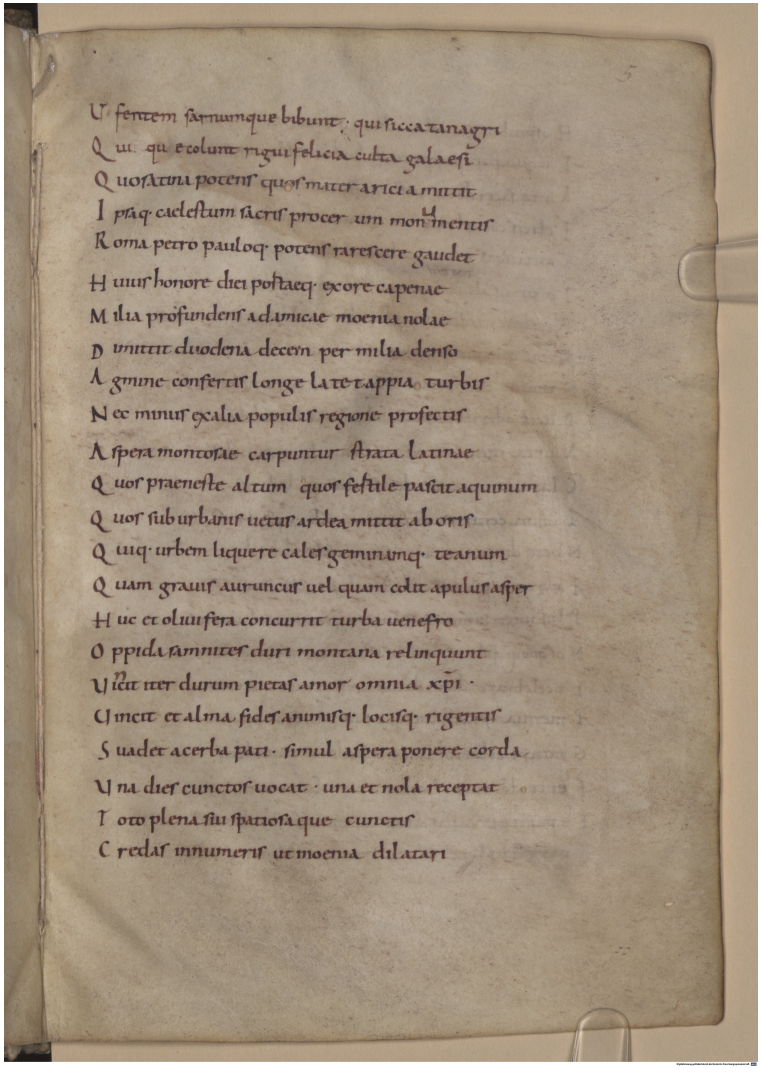
Preprocessing

- **Binarization**: convert grayscale image into a binary image, with each pixel classified as foreground (usually black) or background (usually white), to **create a clear distinction between the text elements and the paper or background** on which the text is written.
- **Scaling**: resize an image to a different size while maintaining its aspect ratio.

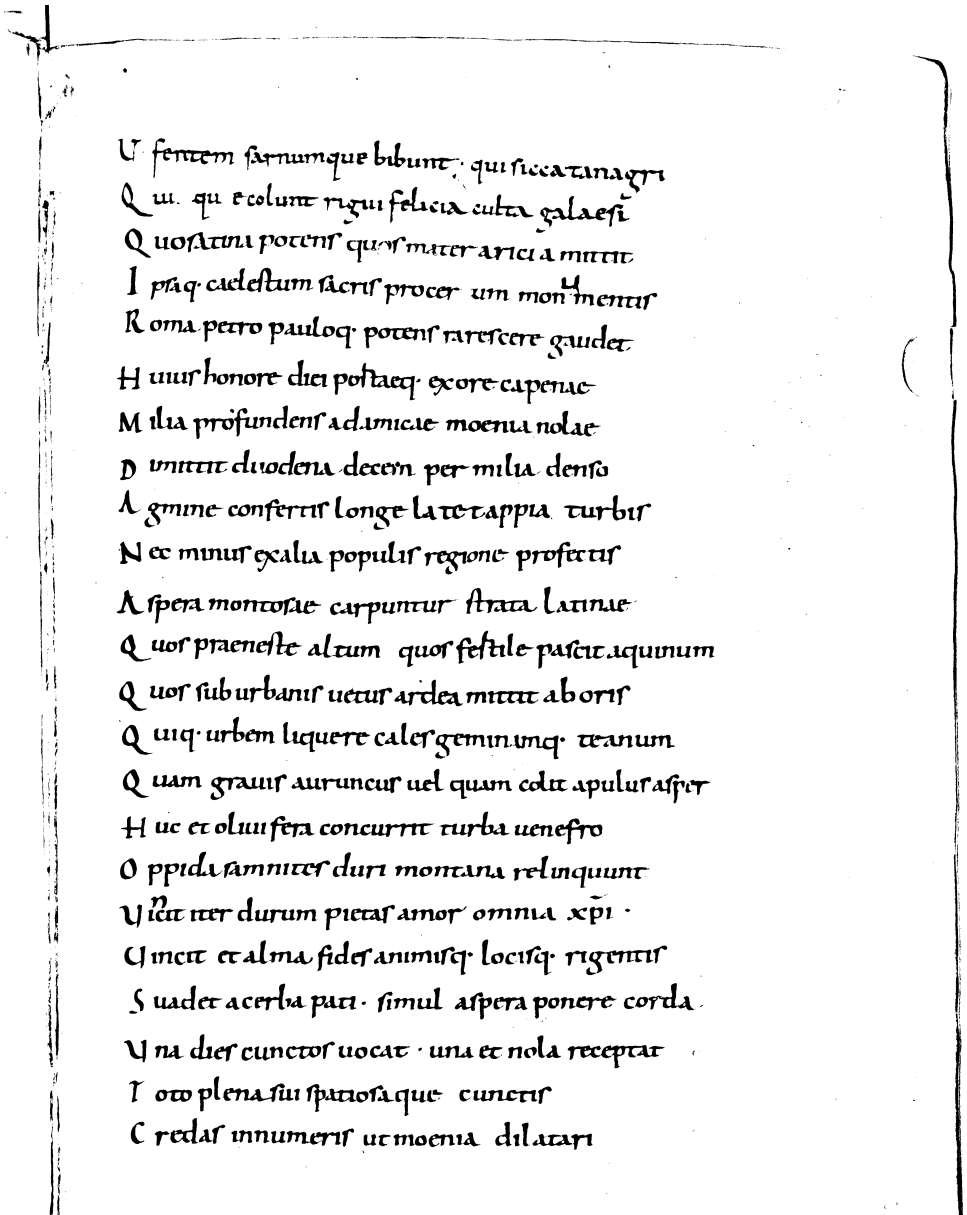
Preprocessing

- **Binarization**: convert grayscale image into a binary image, with each pixel classified as foreground (usually black) or background (usually white), to **create a clear distinction between the text elements and the paper or background** on which the text is written.
- **Scaling**: resize an image to a different size while maintaining its aspect ratio.
- **Noise reduction**: enhance the signal (desired information) and suppress or remove the noise using transformation filters (such as Gaussian blur).

Example of preprocessing



Example of preprocessing



Segmentation

Page is then segmented into

- Zone of texts are detected.
- Lines of text are detected.
- Optionally, individual words/characters are detected.

Line segmentation

Line segmentation

Line segmentation refers to the process of identifying and separating individual lines of text within a document or an image containing multiple lines of text.

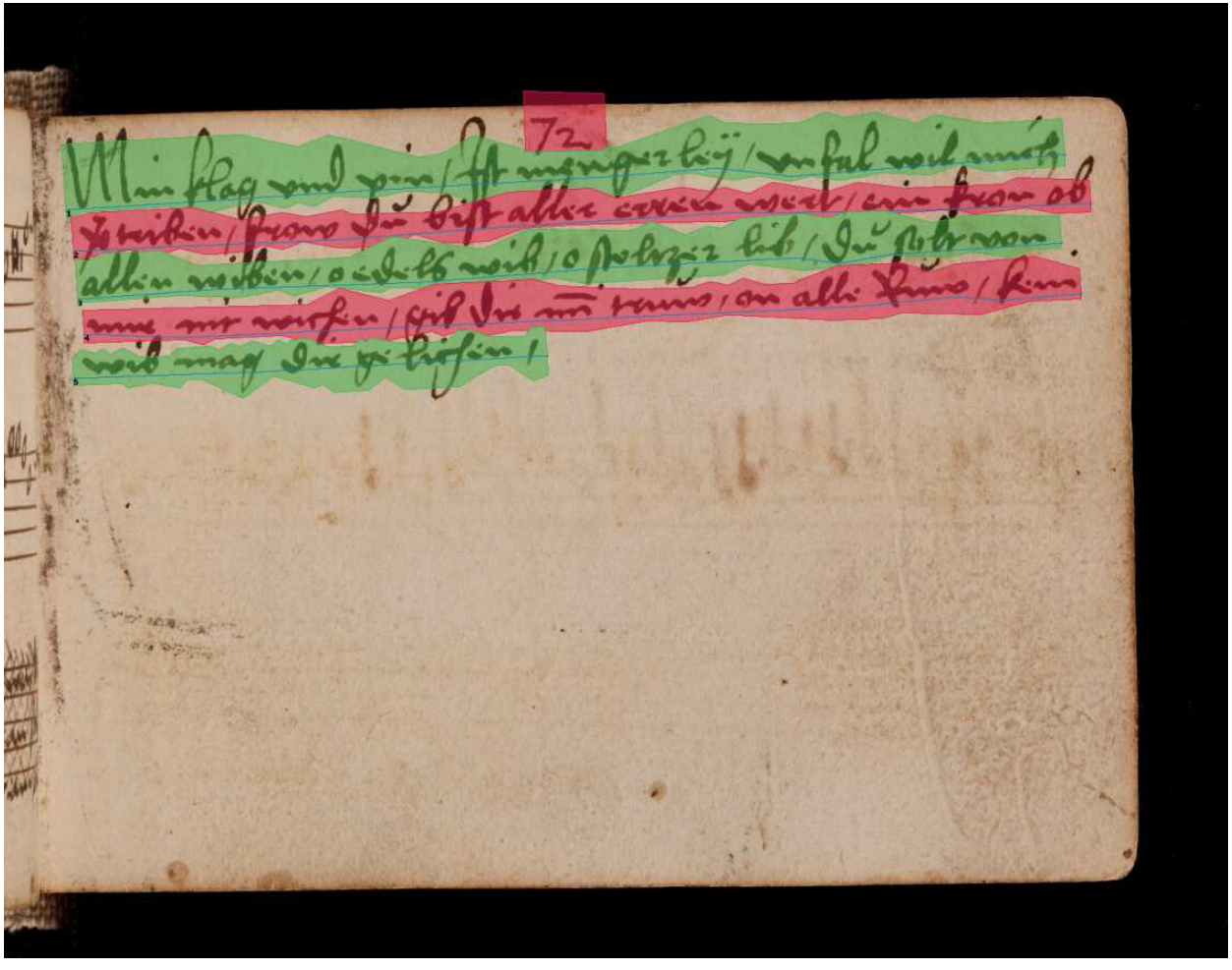
Line segmentation

Line segmentation

Line segmentation refers to the process of identifying and separating individual lines of text within a document or an image containing multiple lines of text.

The goal of line segmentation is to correctly **identify the boundaries between lines of text**, taking into account variations in text alignment, spacing, font size, and formatting

Example of line segmentation



Credit: https://kraken.re/4.3.0/advanced.html

Line segmentation

Line segmentation consists in using a Machine Learning model to:

- label each pixel on the image with one or more classes, each class corresponding to a line or region of a specific type.
- extract line instances and region boundaries, followed by bounding polygon computation for the baselines

Question

What kind of learning is a segmentation model ?

Character segmentation

Additionally, individual characters can be segmented:

Character segmentation

Character segmentation is an operation that seeks to decompose an image of a sequence of characters into sub-images of individual symbols.

Character segmentation

Additionally, individual characters can be segmented:

Character segmentation

Character segmentation is an operation that seeks to decompose an image of a sequence of characters into sub-images of individual symbols.

Given the starting point a document image:

1. Find the next character image.
2. Extract distinguishing attributes of the character image.
3. Find the member of a given symbol set whose attributes best march those of the input.
4. Output the closest match.

Character segmentation

Question

What is the strongest limit to a character based approach ?

Character segmentation

Question

What is the strongest limit to a character based approach ?
Text is considered as a set of unordered characters instead of a sequence of meaningful units.

Recognition models

A supervised model is then fitted to **recognize words**, either on a character per character basis or on a word to word basis.

What is the training data ?

Post-processing

A natural language model is then fitted to correct the result: dictionary match, automatic correction ...

OCR and Deep Learning

OCR requires two supervised learning models:

- A Segmentation Model
- A Recognition Model

These models belong to a particular subclass of **Machine Learning** models called **Deep Learning** models.

Introduction to Deep Learning

Outline

Definition

Deep Learning

Deep learning consists in using a sub-class of machine learning methods, called **artificial neural networks**.

The adjective "deep" in deep learning refers to the use of multiple layers in the network.

Possible applications

Deep learning can be used to solve:

- Supervised problems (classification and regression)
- Unsupervised problems (mostly data projection)

Possible applications

Deep learning can be used to solve:

- Supervised problems (classification and regression)
- Unsupervised problems (mostly data projection)

Deep Learning is especially popular for complex artificial intelligence task, such as *computer vision* and *natural language processing*.

Basic structure: the perceptron

Outline

The Perceptron

Perceptron

A **perceptron** is a type of **binary** linear classifier, as a model of how the human processes information. It is the building block of more complex neural network architectures.

The Perceptron

Perceptron

A **perceptron** is a type of **binary** linear classifier, as a model of how the human processes information. It is the building block of more complex neural network architectures.

If $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are the **input features** and $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ the corresponding **weight vectors**, the perceptron calculates **the weighted sum of inputs and weights**, and then passes it through an **activation function** to make a binary prediction:

$$\text{Output} = \text{activation}\left(\sum_{i=1}^{n} w_i \cdot x_i + w_0\right)$$

Perceptron Architecture

The perceptron consists of the following components:

Perceptron Architecture

The perceptron consists of the following components:

1. **Input Layer**: Receives input features $(x_1, x_2, ..., x_n)$.

Perceptron Architecture

The perceptron consists of the following components:

1. **Input Layer**: Receives input features $(x_1, x_2, ..., x_n)$.
2. **Weights**: Each input feature is associated with a weight $(w_0, w_1, w_2, ..., w_n)$ that determines its importance in the model.

Perceptron Architecture

The perceptron consists of the following components:

1. **Input Layer**: Receives input features $(x_1, x_2, ..., x_n)$.
2. **Weights**: Each input feature is associated with a weight $(w_0, w_1, w_2, ..., w_n)$ that determines its importance in the model.
3. **Summation Function**: Calculates the weighted sum of inputs and weights.

Perceptron Architecture

The perceptron consists of the following components:

1. **Input Layer**: Receives input features $(x_1, x_2, ..., x_n)$.
2. **Weights**: Each input feature is associated with a weight $(w_0, w_1, w_2, ..., w_n)$ that determines its importance in the model.
3. **Summation Function**: Calculates the weighted sum of inputs and weights.
4. **Activation Function**: Decides the output based on the summation result.

Perceptron Architecture

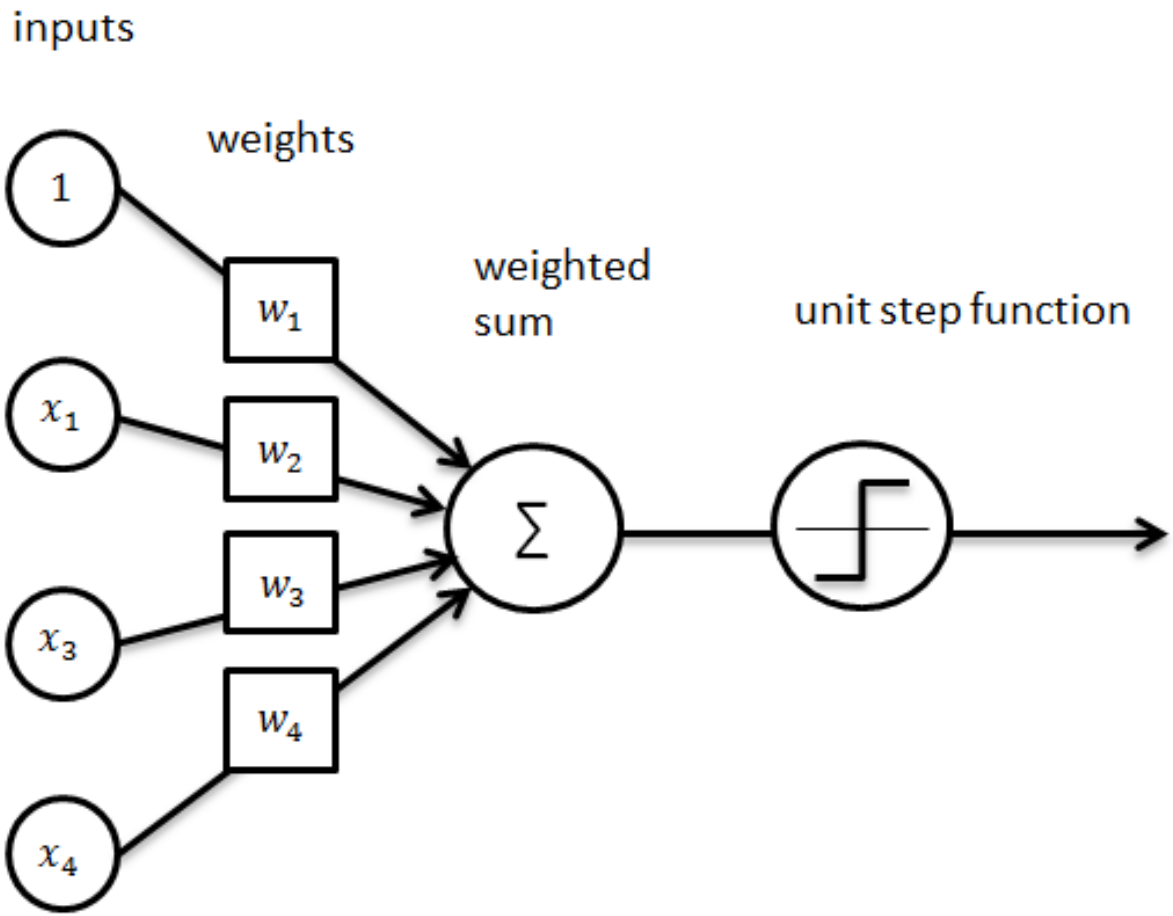
The perceptron consists of the following components:

1. **Input Layer**: Receives input features $(x_1, x_2, ..., x_n)$.
2. **Weights**: Each input feature is associated with a weight $(w_0, w_1, w_2, ..., w_n)$ that determines its importance in the model.
3. **Summation Function**: Calculates the weighted sum of inputs and weights.
4. **Activation Function**: Decides the output based on the summation result.

The perceptron's output is either:

- A classification label (classification)
- A probability distribution (classification)
- An estimated value (regression)

Perceptron Architecture



Activation functions

Popular activation function:

Activation functions

Popular activation function:

- **Simple threshold function**:

$$f(X, W) = \begin{cases} 0, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + w_0 < 0 \\ 1, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + w_0 \geq 0 \end{cases}$$

Activation functions

Popular activation function:

- **Simple threshold function**:

$$f(X, W) = \begin{cases} 0, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + w_0 < 0 \\ 1, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + w_0 \geq 0 \end{cases}$$

- **Sigmoid function**, very popular for classification as it outputs what can be interpreted as a probability distribution

$$f(X, W) = \frac{1}{1 + e^{-(\sum_{i=1}^{n} w_i \cdot x_i + w_0)}}$$

Example

Given a perceptron with weights $(w_0 = 2, w_1 = 3, w_2 = 4)$, predict the binary classification of vector using a linear activation function:

$$[1, 2]$$

and

$$[1, -2]$$

.

Training a perceptron

How do we determine the values of the weights ?

Training a perceptron

How do we determine the values of the weights ?

Neural network training

Determining the values of the different weights is called **training** a neural network, by leveraging the training examples given in the data.

Training a perceptron

How do we determine the values of the weights ?

Neural network training

Determining the values of the different weights is called **training** a neural network, by leveraging the training examples given in the data.

For a single perceptron:

- Initialize weights and bias randomly or with zeros.
- For each training example $(\mathbf{x}, y)$:
  - Compute the predicted output $\hat{y}$ using the current weights.
  - Update the weights based on the error:
    $\Delta w_i = \eta \cdot (y - \hat{y}) \cdot x_i$, with η is the learning rate.

  Repeat until weights do not move anymore or certain number of iterations have been reached.

Training a perceptron

What do you think is the issue with training the perceptron on the whole dataset ?

The algorithm is going to **learn perfectly** the training example but won't be able to generalize on unseen examples.

Overfitting

Overfitting

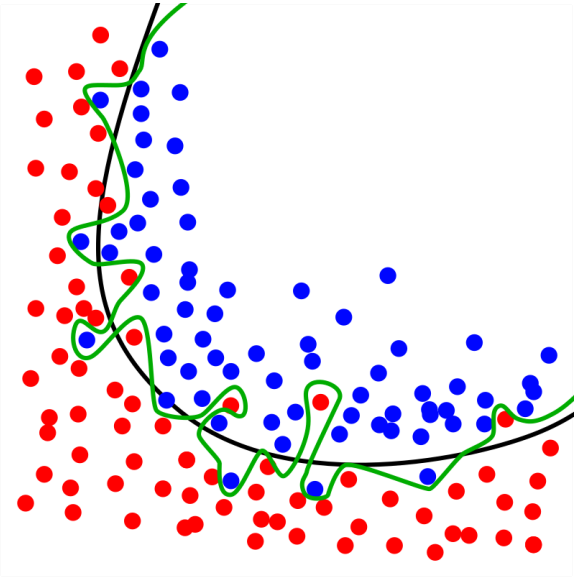
It is always possible to build a function that will match EXACTLY the training dataset, but it doesn't mean it will generalize well ! Learning the noise instead of the global trend in the data leads to **overfitting**.

Overfitting

Overfitting

It is always possible to build a function that will match EXACTLY the training dataset, but it doesn't mean it will generalize well ! Learning the noise instead of the global trend in the data leads to **overfitting**.

It is not always a good idea to have a **perfect fit** for f (i.e, $f(X_i) = y_i$ for all i in the training set).



Training / testing dataset

If a 100% fit on the training set is not always the sign of a good model, how can we select the best model ?

Training / testing dataset

If a 100% fit on the training set is not always the sign of a good model, how can we select the best model ?

Train and test dataset

We split the dataset into two datasets:

- **Training* dataset**: dataset to build the model
- **Testing* dataset**: dataset to test the model (i.e. compute the scoring metric)

Training / testing dataset

If a 100% fit on the training set is not always the sign of a good model, how can we select the best model ?

Train and test dataset

We split the dataset into two datasets:

- **Training* dataset**: dataset to build the model
- **Testing* dataset**: dataset to test the model (i.e. compute the scoring metric)

Question

What issue do you see with this approach ? What should we make sure of when splitting the dataset ?

Evaluation

How can I measure how well the algorithm performs ?

Evaluation

How can I measure how well the algorithm performs ?

Question

Can you give possible evaluation metrics ?

Evaluation

How can I measure how well the algorithm performs ?

Question

Can you give possible evaluation metrics ?

We need objective metric(s) to assess the quality of the model we designed.

Confusion matrix

Confusion* matrix

A **confusion matrix** is a specific table layout that allows visualization of **the performance of a classification algorithm**. Each row of the matrix represents the instances the predicted class while each column represents the instances of the actual class.

Confusion matrix

Confusion* matrix

A **confusion matrix** is a specific table layout that allows visualization of **the performance of a classification algorithm**. Each row of the matrix represents the instances the predicted class while each column represents the instances of the actual class.

The **confusion matrix** makes it easy to see when the algorithm is "confused".

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Example: compute confusion metric

Build the confusion matrix for the following dataset:

Actual class	Predicted class
0	1
0	0
0	0
0	1
1	1
1	1
1	0

Example: compute confusion metric

Build the confusion matrix for the following dataset:

Actual class	Predicted class
0	1
0	0
0	0
0	1
1	1
1	1
1	0

Possible evaluation metrics

Many evaluation metrics rely on the confusion matrix: Precision, Recall, Accuracy, F1-score ...
Because this lecture is quite dense, we will only learn about the accuracy:

Accuracy*: the proportion of true results among the total number of cases examined.
Accuracy $= \frac{TP+TN}{TP+FP+FN+TN}$

Example: compute metric value

Using the previous dataset, compute the accuracy.

Multi-Layer Perceptrons

Outline

Combining perceptrons for feed-forward neural networks

Single layer perceptrons are only able to solve **linearly separable problems**, which is rarely the case in "real-life" data.

Combining perceptrons for feed-forward neural networks

Single layer perceptrons are only able to solve **linearly separable problems**, which is rarely the case in "real-life" data.

To be able to solve non-linear problems, we can stack single perceptron units to create a **network**.

Combining perceptrons for feed-forward neural networks

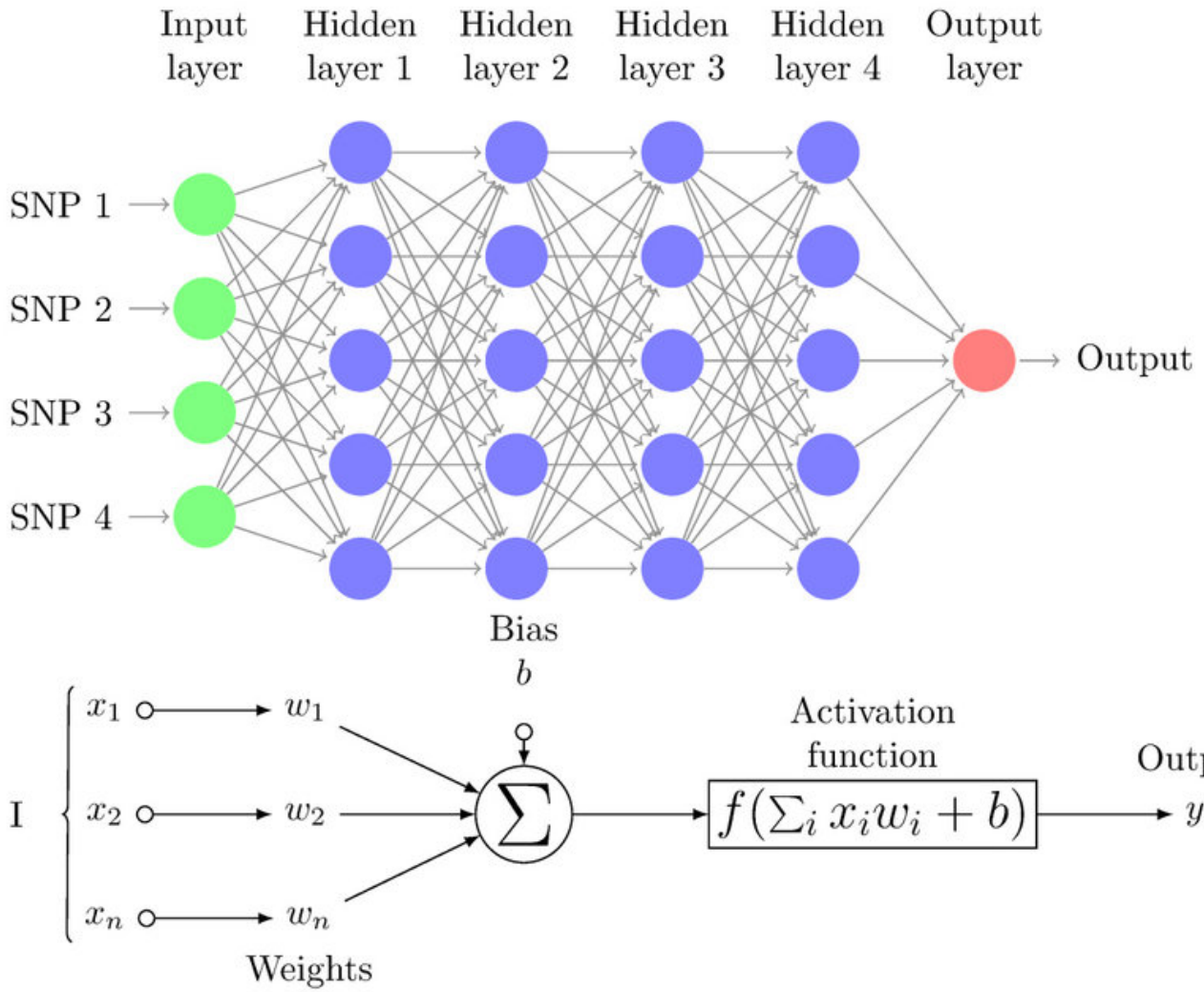
Single layer perceptrons are only able to solve **linearly separable problems**, which is rarely the case in "real-life" data.

To be able to solve non-linear problems, we can stack single perceptron units to create a **network**.

Multi-layer perceptrons

The **multi-layer perceptron (MLP)** is formed by stacking multiple single-layer perceptrons together, making it **deep**. It becomes composed of an **input layer**, **hidden layers** and **output layers**.

Multi-layer perceptrons



Training multi-layer perceptron

To train the neural network, we need some kind of measurement to "teach" it, which is called **loss**. A popular loss function is **cross-entropy**.

Training multi-layer perceptron

To train the neural network, we need some kind of measurement to "teach" it, which is called **loss**. A popular loss function is **cross-entropy**.

Cross-entropy

Cross entropy measures the similarity between two distributions. The formula for cross-entropy between two discrete probability distributions P and Q is given by:

$$H(P, Q) = -\sum_{x \in \mathcal{X}} P(x) \log Q(x)$$

where:

- $P(x)$ is the probability of event x according to distribution P.
- $Q(x)$ is the probability of event x according to distribution Q.

Cross entropy

In the case of binary classification, if y_i is the correct label and $p_{i,j}$ the estimated probability returned by the neural network (on which we will base prediction $\hat{y}_{i,j}$) then the cross-entropy for example i defined as:

$$CE(y_i, \hat{y}_i) = -\sum_{j=1}^{n} y_{i,j} \cdot \log(p_i) = -y_i log(p_i) - (1 - y_i) \times log(1 - p_i)$$

Cross-entropy is then summed over all classification tasks.

Cross-entropy

Question

Compute the cross-entropy and the accuracy (p threshold of 0.5) for the following classification results:

- Ground truth vector: $y = [1, 1, 0, 1]$
- Probability vector $p = [0.3, 0.9, 0.2, 0.8]$

Gradient descent

To obtain the final network, we need to **infer all the weights of all the layers**.

Gradient descent

To obtain the final network, we need to **infer all the weights of all the layers**.
The cross-entropy is a **non-derivable** function of the parameters, which optimum cannot be expressed in a closed form formula: it needs to be **numerically computed**, using an algorithm called **back-propagation**.

Gradient descent

To obtain the final network, we need to **infer all the weights of all the layers**.
The cross-entropy is a **non-derivable** function of the parameters, which optimum cannot be expressed in a closed form formula: it needs to be **numerically computed**, using an algorithm called **back-propagation**.
The intuition behind **back-propagation** (we will not dwelve into the math) is that **each weight contributes a little to the error of the network**.

Recurrent Neural Networks and LSTM

The neural networks architecture we just discovered was **fast-forward architecture**: each layer is fully connected and data only goes **forward**.

Recurrent Neural Networks and LSTM

The neural networks architecture we just discovered was **fast-forward architecture**: each layer is fully connected and data only goes **forward**.

Question

Does the input matrix necessarily has the same size with feed-forward neural network ? Is this the case when working with unstructured data, such as text ?

Recurrent Neural Networks

Recurrent Neural Network

Recurrent Neural Network are a class of artificial neural networks designed to handle sequential data by **introducing a feedback loop that allows information to persist**.

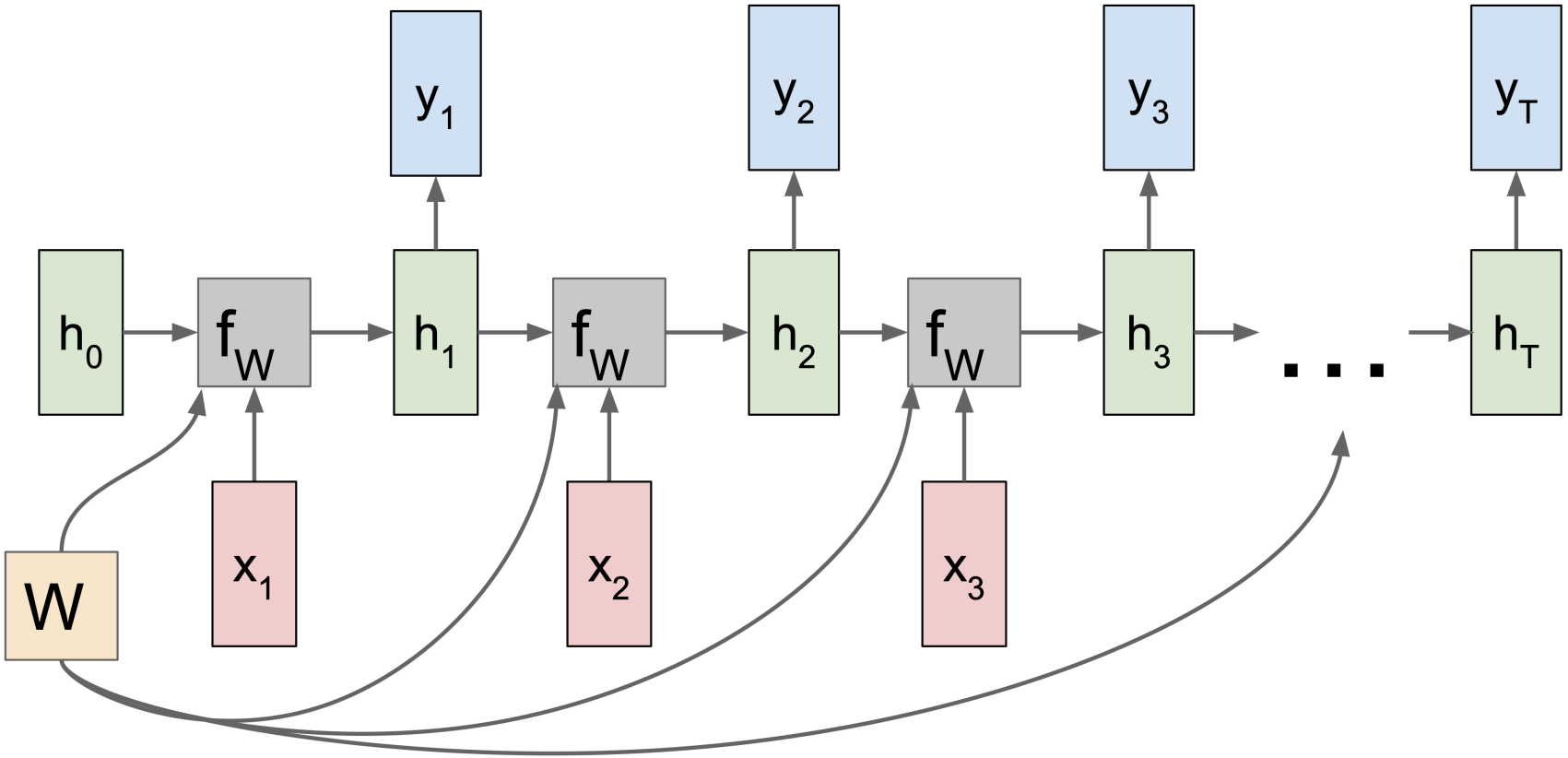
Recurrent Neural Networks

Recurrent Neural Network

Recurrent Neural Network are a class of artificial neural networks designed to handle sequential data by **introducing a feedback loop that allows information to persist**.

They possess internal memory **which enables them to maintain a hidden state**, updated at each time step, which is a function of **the hidden state** and **the current input**.

Recurrent Neural Networks



Credit: Stanford Lecture 10 on Neural Networks for computer vision (2017)

Recurrent Neural Networks

RNN Architecture

$$h_t = f_W(h_{t-1}, x_t) = tanh(W_{in}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{out}h_t$$

Where:

- h_t is the hidden state at time step t.
- x_t is the input at time step t.
- W_{in} is the input-to-hidden weight matrix.
- W_{out} is the hidden-to-output weight matrix.
- W_{hh} is the hidden-to-hidden weight matrix.
- b_h is the bias vector for the hidden layer.
- b_y is the bias vector for the output layer.

Long Term Short Term Network

The major issue with standard RNN are:

- **Vanishing/Exploding Gradients:** When backpropagating gradients through multiple time steps and computing multiple tanh, the gradients can
  - become exponentially small (largest singular value lt 1)
  - become exponentially big (largest singular value gt 1)
- **Short-term Memory:** difficulty retaining information over long sequences.

Long Term Short Term Network

The major issue with standard RNN are:

- **Vanishing/Exploding Gradients:** When backpropagating gradients through multiple time steps and computing multiple tanh, the gradients can
  - become exponentially small (largest singular value lt 1)
  - become exponentially big (largest singular value gt 1)
- **Short-term Memory:** difficulty retaining information over long sequences.

In 1997, **a new neural network architecture has been introduced**, called **Long Short Term Memory** neural networks. It has since then **become OMNIPRESENT** in Deep Learning.

The LSTM architecture

LSTM

LSTM (Hochreiter and Schmidhuber, 1997) is a type of Recurrent Neural Network which **utilizes gating mechanisms to control the flow of information**, enabling better memory retention and gradient flow during training.

The LSTM architecture

LSTM

LSTM (Hochreiter and Schmidhuber, 1997) is a type of Recurrent Neural Network which **utilizes gating mechanisms to control the flow of information**, enabling better memory retention and gradient flow during training.

It introduces the concept of **cell state** c_t, with:

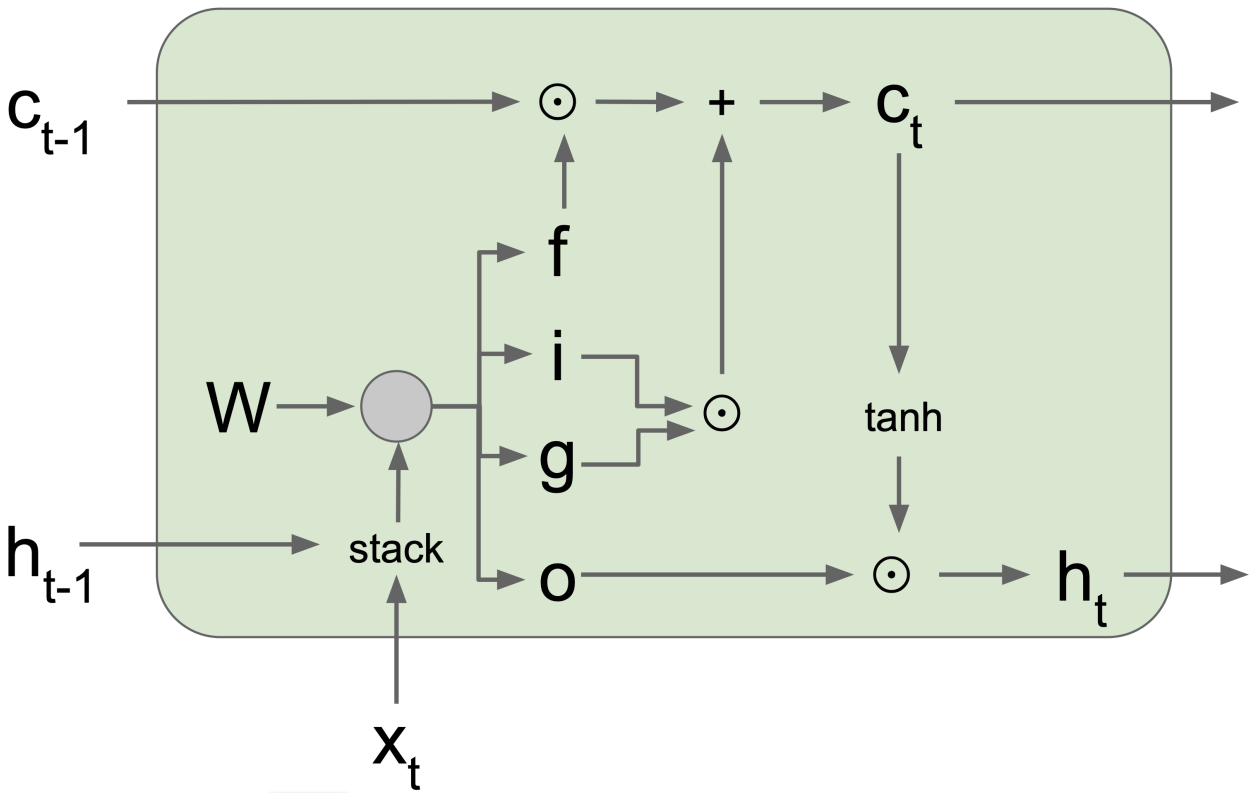
- **Forget Gate (f_t):** Determines what information to discard from the previous cell state.
- **Input Gate (i_t):** Determines what new information to store in the cell state.
- **Output Gate (o_t):** Controls the flow of information from the current cell state to the hidden state and the output.

LSTM Computation

Given the input at time step t (x_t), the previous hidden state (h_{t-1}), and the previous cell state (c_{t-1}), LSTM compute:

Concatenated Input:	$\mathbf{x}_t = [h_{t-1}, x_t]$
Forget Gate:	$\mathbf{f}_t = \sigma(W_f \cdot \mathbf{x}_t + b_f)$
Input Gate:	$\mathbf{i}_t = \sigma(W_i \cdot \mathbf{x}_t + b_i)$
Candidate Cell State:	$\tilde{\mathbf{c}}_t = \tanh(W_c \cdot \mathbf{x}_t + b_c)$
Updated Cell State:	$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$
Output Gate:	$\mathbf{o}_t = \sigma(W_o \cdot \mathbf{x}_t + b_o)$
Updated Hidden State:	$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$
Output at Time Step t:	$y_t = \text{Activation}(W_{out} \cdot \mathbf{h}_t + b_y)$

The LSTM cell



Benefits of LSTM

- **Long-Term Dependencies:** Long term data dependencies are retained, as the gating mechanisms allow the network to control the flow of information into and out of the cell state.

Benefits of LSTM

- **Long-Term Dependencies:** Long term data dependencies are retained, as the gating mechanisms allow the network to control the flow of information into and out of the cell state.
- **Vanishing Gradient Mitigation:** The gating mechanism enables effective learning by addressing the vanishing and exploding gradient problems in vanilla RNNs by removing the w factor in the gradient computation.

Lab session

Outline

About me

- Master's degree in Applied Mathematics;
- Technical leader at a start-up in Grenoble;
- Work on next-gen quality detection using multiple sensor data analysis and computer vision.

Lab purpose

The lab is separated into two parts:

Lab purpose

The lab is separated into two parts:

1. Use Kraken to transcribe manuscript data.

Lab purpose

The lab is separated into two parts:

1. Use Kraken to transcribe manuscript data.
2. Train our own Deep Learning model for manuscript transcription.

Kraken

Kraken

Kraken* is an open-source Optical Character Recognition (OCR) engine designed for processing historical and archival documents.

Kraken

Kraken

Kraken* is an open-source Optical Character Recognition (OCR) engine designed for processing historical and archival documents.

Key Features:

- **Ready to use**: Out of the box highly accurate trained models on diverse scripts and languages, suitable for historical texts.

Kraken

Kraken

Kraken* is an open-source Optical Character Recognition (OCR) engine designed for processing historical and archival documents.

Key Features:

- **Ready to use**: Out of the box highly accurate trained models on diverse scripts and languages, suitable for historical texts.
- **Line Recognition**: Designed to handle complex layouts and historical handwriting.

Kraken

Kraken

Kraken* is an open-source Optical Character Recognition (OCR) engine designed for processing historical and archival documents.

Key Features:

- **Ready to use**: Out of the box highly accurate trained models on diverse scripts and languages, suitable for historical texts.
- **Line Recognition**: Designed to handle complex layouts and historical handwriting.
- **Model Flexibility**: Supports training custom models for specific data.

Basic workflow with Kraken

Can you remind me of the OCR steps we saw this morning ?

Basic workflow with Kraken

Can you remind me of the OCR steps we saw this morning ?

- Preprocess (binarize, scale, filter noise, rotate);
- Segment the text on a line;
- Textual recognition using Deep Neural Networks;
- Manual/Automatic validation of results

Basic workflow with Kraken

Can you remind me of the OCR steps we saw this morning ?

- Preprocess (binarize, scale, filter noise, rotate);
- Segment the text on a line;
- Textual recognition using Deep Neural Networks;
- Manual/Automatic validation of results

Kraken does most of this for you !

Basic workflow with Kraken

1. **Preprocessing**:
   - Prepare scanned images of historical documents.
   - Clean images by removing noise, enhancing contrast, and adjusting resolution.

Basic workflow with Kraken

1. **Preprocessing**:
   - Prepare scanned images of historical documents.
   - Clean images by removing noise, enhancing contrast, and adjusting resolution.
2. **Prepare model**:
   - Download models from a supported collaborative platform such as Zenodo.
   - If required, train a custom model for specific scripts or languages using Kraken's training tools and an annotating tool.

Basic workflow with Kraken

1. **Preprocessing**:
   - Prepare scanned images of historical documents.
   - Clean images by removing noise, enhancing contrast, and adjusting resolution.
2. **Prepare model**:
   - Download models from a supported collaborative platform such as Zenodo.
   - If required, train a custom model for specific scripts or languages using Kraken's training tools and an annotating tool.
3. **Recognition**:
   - For each image, use Kraken's command line to extract text.
   - Kraken's neural networks recognizes the text and outputs human-readable or machine-readable text for each image.

Basic workflow with Kraken

1. **Preprocessing**:
   - Prepare scanned images of historical documents.
   - Clean images by removing noise, enhancing contrast, and adjusting resolution.
2. **Prepare model**:
   - Download models from a supported collaborative platform such as Zenodo.
   - If required, train a custom model for specific scripts or languages using Kraken's training tools and an annotating tool.
3. **Recognition**:
   - For each image, use Kraken's command line to extract text.
   - Kraken's neural networks recognizes the text and outputs human-readable or machine-readable text for each image.
4. **Validation**: Correct any recognition errors, and adapt and re-evaluate your workflow accordingly.

Installation

Kraken is delivered as a Python package:

Question

Do you remember how to install a Python package ?

Installation

Kraken is delivered as a Python package:

Question

Do you remember how to install a Python package ?

In your virtual environment:

```
pip install kraken
```

or add Kraken to your project's dependencies.

List all available models and download one

Pre-trained models

Kraken supports various pre-trained models for different scripts and languages:

- **Script-specific models**: trained on specific characters such as Latin, Cyrillic, Hebrew, Arabic ...
- **Historical models**: trained on fonts and scripts from specific historical periods.
- **Handwritten model**: trained to handle for a specific handwriting ...

Listing available models

To list all available models:

```
kraken list
```

Listing available models

To list all available models:

```
kraken list
```

Select the wanted model. For the lab, we will use:

```
kraken get "10.5281/zenodo.7631619"
```

This will download the model into a directory depending on your operating system.

Numerizing the content in the command line

You can run every step that we saw in the lecture at once using:

```
kraken \
  -i demo_data/demo_image.jpg \
  output.txt \
  segment \
  -bl ocr \
  -m cremma-generic-1.0.1.mlmodel
```

Data taken from

`https://github.com/rescribe/carolineminuscule-groundtruth`.

Analyzing the results

Go to the jupyter notebook provided with the lab session to analyze the output of Kraken.

Training our own OCR model

The out of the box models made available by Kraken can be not enough for your particular project.

Can you remind me from this morning lecture what is required to obtain a new character recognition predictor ?

Training our own OCR model

The out of the box models made available by Kraken can be not enough for your particular project.

Can you remind me from this morning lecture what is required to obtain a new character recognition predictor ?

To obtain our own predictor we need:

- A Deep Learning Model architecture in VGSL;
- Training data in the Alto XML format;
- A Python virtual environment with Ketos installed.

Training our own OCR model

VGSL specification

Variable-size Graph Specification Language (VGSL) enables the specification of a neural network, composed of convolutions and LSTMs, that can process variable-sized images, from a **compact definition string**.

In one string, one can specify the architecture of a neural network that will then be used for training.

Training our own OCR model

Example of VGSL string:

`1,0,0,3 Ct5,5,16 Mp3,3 Lfys64 Lfx128 Lrx128 Lfx256 O1c105`

- First four digits:

  batch, *height*, *width*, *depth*

  .
  - Batch is ignored.
  - Height/Width set to 0 means variable height and width.
  - Depth is 1 for greyscale, 3 for color processing.
- Last "word" describes output layer
- Middle "words" correspond to the wanted architecture of the network: convolutions, LSTM ...

Documentation is available at:

`https://tesseract-ocr.github.io/tessdoc/tess4/VGSLSpecs.html`

Training the data: The ALTO XML format

ALTO

ALTO (Annotation on Language Toolkit) XML is a markup language used for **representing ground truth data in document analysis and recognition tasks**. It provides a standardized way to represent the layout and content of documents.

ALTO XML data holds the layout and content information to create input-output pairs required to train Deep Learning models.

The ALTO XML format

Example of ALTO specification is located in the demo_data folder.

Example of ALTO transcription projects

Non-exhaustive list:

- `https://github.com/HTR-United` (great website with a research motor: `https://htr-united.github.io/index.html#top`)
- `https://github.com/Gallicorpora`
- `https://github.com/rescribe`

Training our own model using Kraken

Training data is included in the folder: `training_data`.
Because training data can be very long, **we have selected a small data sample** (IRL, you'll want to sample a lot more data !).

Data source:

`https://github.com/rescribe/carolineminuscule-groundtruth`.

Training our own model using Kraken

Training a Kraken model

Training Kraken models use the `ketos` command.

`keros train` trains your model, specifying training data and a network architecture using the -s parameter. **Go experiment it in the lab session** !

More advanced usage can consist in refining or slicing an existing model: see `https://kraken.re/4.3.0/ketos.html`.

Testing the accuracy of the model

As we performed manually in the Jupyter Notebook, we can also use ketos to test the performance of an existing model:

- A pre-trained model
- Our own trained model.

Syntax is:

```
ketos test -m $model -e ../demo_data/ ../demo_data/*.jpeg
```

Generating your own data

Kraken allows to generate the parsed data in the Alto XML format (ideal for a quick touch-up) using the following command:

```
kraken -a -i "../demo_data/demo_image.jpeg" output.txt
  segment -bl
  ocr -m cremma-generic-1.0.1.mlmodel
```

UI to generate data

Kraken does not natively handle ALTO XML data generation, but several tools are available:

- Transkribus (`https://readcoop.eu/transkribus/`)
- Aletheia (`https://www.primaresearch.org/tools/Aletheia`)
- e-Scriptorium (`https://msia.escriptorium.fr`)
-

Questions

Questions ?