MINES PARIS | PSL | Centre de recherche en informatique

Traitement Automatique des Langues
Natural Language Processing
Notes de cours

Georges-André Silber, CRI, Mines Paris, Université PSL
georges-andre.silber@minesparis.psl.eu
https://mines.paris/nlp

ES3A_MES-07, 2025–2026
Version du 15 décembre 2025

Table des matières

1 Logistique du cours

Le cours de Traitement Automatique des Langues (TAL), en anglais *Natural Language Processing* (NLP), se compose de 16 séances d'1h30. Ces séances seront partagées entre cours magistraux et travaux pratiques, principalement en Python 3.

Votre note finale au cours sera la moyenne pondérée des notes obtenues pour chaque TP que vous me rendrez et du projet final, ce dernier ayant une pondération de 1/2. Un principe général est que vous me rendiez tous les TP, avec un joker possible. Les TP seront à rendre en fin de séance, avec la possibilité de le rendre plus tard, mais impérativement avant la séance suivante.

Les moyens de calcul que vous pourrez utiliser pour les travaux pratiques et le projets seront composés de vos propres machines, de machines des mines et de moyens de calculs loués chez Scaleway pour ce cours.

Vous aurez le choix entre plusieurs projets (6 dans l'édition 2024–2025 du cours), avec un niveau technique différent entre chaque projet.

2 Introduction

2.1 L'IA dans la pop culture

Le développement récent de l'IA et du NLP est une révolution culturelle, notamment depuis le "choc" chatGPT de 2022. L'IA a cependant une longue histoire, notamment dans le domaine de la langue, que l'on peut illustrer par des œuvres des la "pop culture" :

— Le "*turc mécanique*" (https://fr.wikipedia.org/wiki/Turc_mÃľcanique) ;
— Le *test de Turing* (Computing Machinery and Intelligence) [6] ;
— *Blade Runner* (Ridley Scott, 1982). Adaptation du livre "*Do Androids Dream of Electric Sheeps ?*" de Philip K. Dick (1966). Terre dévastée en 2019, à Los Angeles, il reste des humains qui n'ont pas pu ou pas choisi d'aller sur mars. Test Voight-Kampff pour détecter les *réplicants*, adaptation du test de Turing ;
— "*2001 l'odyssée de l'espace*" de Stanley Kubrick (1968) et son IA HAL 9000 : https://youtu.be/ARJ8cAGm6JE.
— "*Terminator*" de James Cameron (1984). IA militaire Skynet qui a détruit la planète. Androïde T-800 qui est renvoyé dans le passé pour détruire la mère du futur leader de la resistance (Sarah Connor). https://www.youtube.com/watch?v=QaagRs5pX_E
— "*Wargames*" de John Badham (1984). IA militaire WOPR (War Operation Plan Response), conçue pour pallier la défaillance des humains dans la décision de déclenchement du feu nucléaire.https://youtu.be/7R0mD3uWk5c, https://youtu.be/tGNBdjVO04Y, https://youtu.be/F7qOV8xonfY.
— "*Her*" de Spike Jonze (2013).

2.2 Qu'est-ce que le TAL ?

Fondamentalement, il s'agit d'apprendre les langues aux machines.

Applications du TAL (transparents MVA 2024, de 8 à 27) (transparents originaux).

Challenges (transparents MVA 2024, de 63 à 87).

Les grands modèles de langues réalisent très bien la plupart de ces tâches (les LLM sont à l'état-de-l'art, souvent abrégé SoTA comme *State of The Art*).

2.3 Histoire sélective du TAL

La langue humaine est cœur de l'*Intelligence Artificielle* : "*reproduire (imiter) informatiquement des comportements qui font traditionnellement appel à l'intelligence humaine*". L'utilisation du langage est l'un de ces comportements.

— 1933, les machines à traduire de Georges Artsrouni
— 1940–1949, progrès en théorie des automates, langages formels, probabilités, théorie de l'information. Travaux de Booth, Weaver, Richens
— 1949, Mémorandum "Translation" de Warren Weaver
— 1950, *Computing Machinery and Intelligence* (A. Turing)
— 1954, expérience Georgetown-IBM, traduction du russe vers l'anglais
— 1955, introduction du terme "*Artificial Intelligence*" par John Mac Carthy à la conférence de Dartmouth
— 1958, "The History and Recent Progress of Machine Translation" par A.D. Booth
— 1966, ELIZA (Joseph Weizenbaum)
— 1968, SHRDLU (PhD de Terry Winograd au MIT)
— 1970–2000, « ontologies conceptuelles », approches symboliques
— 1988, approches par apprentissage automatique statistique
— 2010, approches par apprentissage automatique neuronal
— 2018, BERT (Google)
— 2020, GPT-3 (OpenAI)
— 2022, ChatGPT (OpenAI)
— 2023, Poids et code ouvert : llama (Meta)
— 2025, Multimodalité : Gemini (Google), Mistral AI, Claude (Anthropic), GPT 5 (OpenAI)

Memorandum "translation" (1949) Grand impact scientifique et politique :

1. l'ambiguïté peut-être résolue grâce au contexte ;
2. la traduction a une solution formelle, ou solution mécanique, car le langage a une structure logique ;
3. les méthodes cryptographiques s'appliquent (par exemple, l'anglais peut-être vu comme du russe chiffré) ;
4. le sens peut-être représenté indépendamment de la langue.

Expérience Georgetown-IBM (1954) 60 phrases traduites du russe en anglais. 250 mots et 6 règles syntaxiques. Enthousiasme considérable : "*five, perhaps three years*" (IBM), "*dans 15 ans on estime que des traducteurs électroniques pourront être utilisés dans les assemblées internationales comme les nations unies*" (Le Monde). "The Brain" dans les articles de presse. Quelques retours plus nuancés : "*a vast amount of work is still needed*" (Neil Macdonald, 1954), "*a* kitty hawk *flight* (J. Hutchins, 2006).

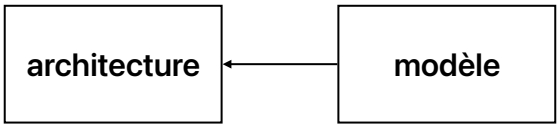
Conférence de Dartmouth (1955) John McCarthy est également l'inventeur du temps partagé, du langage fonctionnel LISP (LISt Processing), prix Turing 1971. L'un des pères fondateurs de la discipline. Le terme "*Artificial Intelligence*" a été utilisé pour la première fois dans le cadre de la session de travail de Dartmouth, connue comme le *Darthmouth Workshop*, où pendant huit semaines se sont réunis McCarthy, Minsky, Rochester et Shannon : "*the study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.*"

2.4 Apprentissage automatique statistique

Deux familles d'approches pour faire faire un calcul à une machine :

— règles (approche symbolique) : on écrit un programme explicite, fixe, décrivant la manière exacte d'effectuer les opérations ;
— apprentissage automatique supervisé : on montre des exemples à la machine du résultat à obtenir, et la machine apprend à reproduire le résultat (trouve elle-meme un "programme" réalisant le calcul).

En apprentissage automatique, on a une architecture fixe et un modèle variable qui comporte des paramètres, tel que décrit dans le schéma ci-dessous.



"Apprendre" → Trouver les "meilleures" valeurs possibles pour les paramètres. Les paramètres du modèles sont chargés dans l'architecture pour reproduire les résultats appris.

2.4.1 Analyse syntaxique statistique

Supposons que l'on cherche à traduire du français vers l'anglais : on veut trouver la meilleure traduction y en anglais d'une phrase x en français. On modélise la qualité d'une traduction par une probabilité $\mathrm{P}(y|x)$. On cherche la meilleure traduction y, que l'on modélise par $\mathrm{argmax}_y \mathrm{P}(y|x)$. L'application du théorème de Bayes, en faisant l'hypothèse raisonnable que $\mathrm{P}(x)$ n'est pas nul puisque c'est la phrase de départ :

$$\mathrm{P}(y \mid x) = \frac{\mathrm{P}(x \mid y).\mathrm{P}(y)}{\mathrm{P}(x)}$$

nous permet de décomposer cette probabilité en deux composantes séparées :

$$\mathrm{argmax}_y \mathrm{P}(x \mid y).\mathrm{P}(y)$$

en ne s'intéressant qu'aux composantes où intervient y.

On a deux modèles, un *modèle de traduction* $\mathrm{P}(x \mid y)$, qui modélise comment les mots et séquences de mots doivent être traduits pour préserver le sens (appris à partir de corpus parallèles), et un *modèle de langue* ($\mathrm{P}(x)$) qui modélise comment produire des phrases en anglais correct (appris à partir de corpus monolingues).

2.4.2 Modèle de langue : distribution de probabilité

Un modèle de langue est une distribution de probabilité sur les séquences de mots, plus la probabilité est élevée, plus la séquence de mots est correcte par rapport à la langue considérée.

Par exemple, en français, il faut que dans notre modèle l'inégalité suivante soit vraie :

$$\mathrm{P}(\text{Je mange une pomme verte}) > \mathrm{P}(\text{est llkdef bla topaz})$$

2.4.3 Création d'un modèle de langue par comptage : n-grams

Pour plus de simplicité, nous allons introduire ici le terme *token* par lequel nous allons désigner indifféremment un caractère, un groupe de caractères ou un mot.

Un n-gram est une séquence de n tokens : un 2-gram ou *bigram* est une séquences de 2 tokens, un 3-gram ou *trigram* une séquence de trois caractères, etc. L'un des modèles de langue les plus simple est le *modèle n-gram* qui est un modèle probabiliste qui peut estimer la probabilité d'un token étant donné les $n-1$ tokens précédents.

Considérons la tâche de déterminer la probabilité $P(t \mid c)$ d'un token t étant donné un *contexte* c, où c est une suite de tokens. Par exemple, avec des tokens équivalents à des mots, considérons que c est égal à La plus belle ville du monde est et qu'on veille connaître la probabilité que t soit Périgueux :

$$P(\text{Périgueux} \mid \text{La plus belle ville du monde est})$$

.

Une manière d'estimer cette probabilité est de compter dans un très grand corpus de textes la fréquence d'apparition f_c de la phrase de contexte "La plus belle ville du monde est" et la fréquence d'apparition f_{ct} de la de la phrase complète "La plus belle ville du monde est Périgueux". On pourrait ensuite déterminer $P(t \mid c) = f_{ct}/f_c$.

Outre que cette approche nécessite une grande quantité de mémoire pour stocker les phrases associées à leurs fréquences, le langage est très divers et une phrase qui n'a pas été vue dans le corpus d'apprentissage donnera une probabilité nulle.

En partant de l'hypothèse que l'on peut approximer le contexte par uniquement quelques tokens précédents, on peut utiliser une chaîne de Markov, permettant par exemple de construire un modèle 3-gram permettant d'écrire, avec le caractère □ indiquant le mot vide :

$$\begin{aligned} &P(\text{La plus belle ville du monde est Périgueux}) \\ &= P(\text{La} \mid □\,□) \times P(\text{plus} \mid □\,\text{La}) \times P(\text{belle} \mid \text{La plus}) \times P(\text{ville} \mid \text{plus belle}) \times \cdots \times P(\text{Périgueux} \mid \text{monde est}) \end{aligned}$$

Le comptage se simplifie car il revient à ne plus compter que des 3-gram pour obtenir des probabilités. Par contre cette approche a des limites dans les contextes long. Comment par exemple déterminer avec le "long" contexte suivant :

"Marie, pour réfléchir, à l'habitude de se parler à"

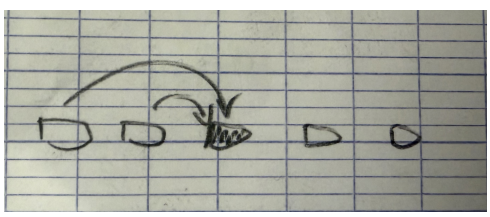
si le token suivant doit être "elle-même" ou "lui-même" ?

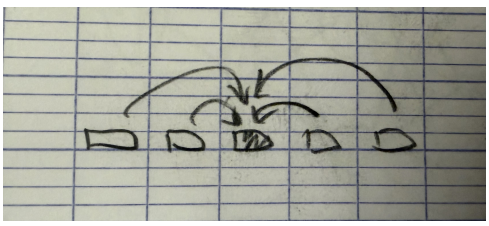
Note : fin de la séance du 9/12/2025.

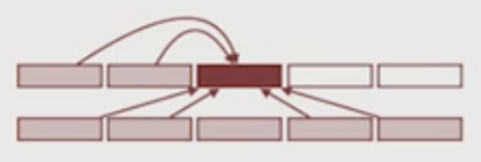
2.4.4 Approches par prédiction

Apprendre à prédire le token le plus probable étant donné un certain contexte :

— contexte gauche : modèle génératif (GPT en apprentissage neuronal)
— contexte complet : modèle par masquage (BERT en apprentissage neuronal)







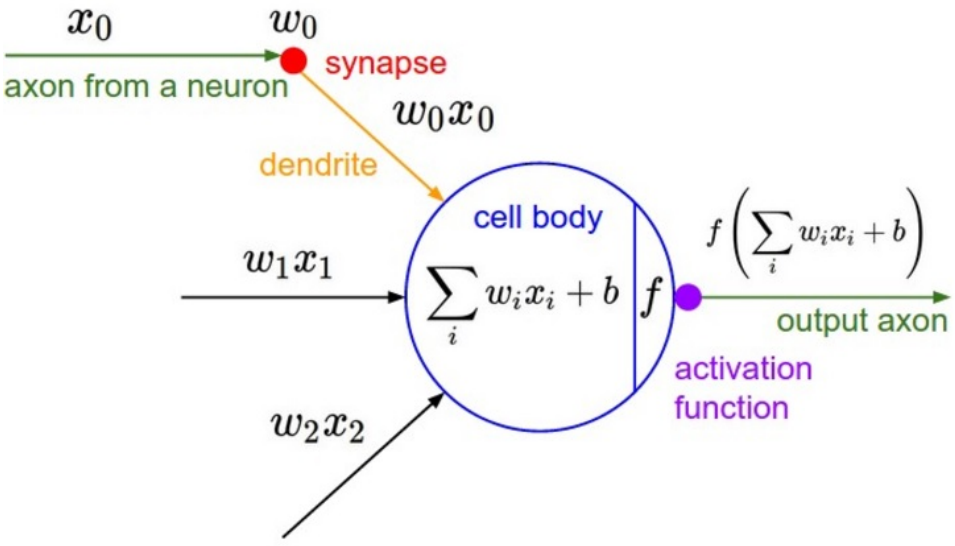
2.5 Apprentissage automatique neuronal

Aujourd'hui : domination des approches neuronales. Pourquoi ? Progression de la recherche (deep learning) ; technologies de la puissance de calcul (cpu, gpu, mémoire, réseaux rapides) ; données massives disponibles sur étagère (internet) ; grands corpus arborés créés à la main (héritage de la linguistique "classique" et statistique).

Histoire sélective et rapide des réseaux neuronaux

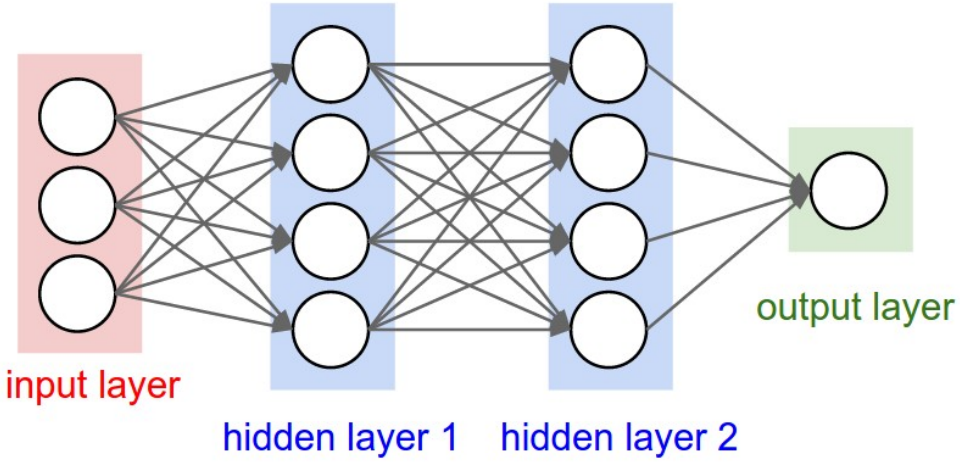
— 1943, Notion de neurone artificiel (McCulloch & Pitts)
— 1957/1958, Apprentissage supervisé, Perceptron (Rosenblatt, 1957, 1958)
— 1962, Plusieurs couches en propagation avant (Rosenblatt)
— 1986, Rétropropagation du gradient (Rumelhart, Hinton, Williams)
— 1989, Réseaux convolutifs (Le Cun *et al.*)
— 1990, Réseaux récurrents (Elman)
— 1997, LSTM (Hochreiter)
— 2006, *Deep Learning*, $c \geq 3$ (Hinton, Bengio)
— 2017, Architecture *Transformer* (Vaswani *et al.*)

Neurone artificiel



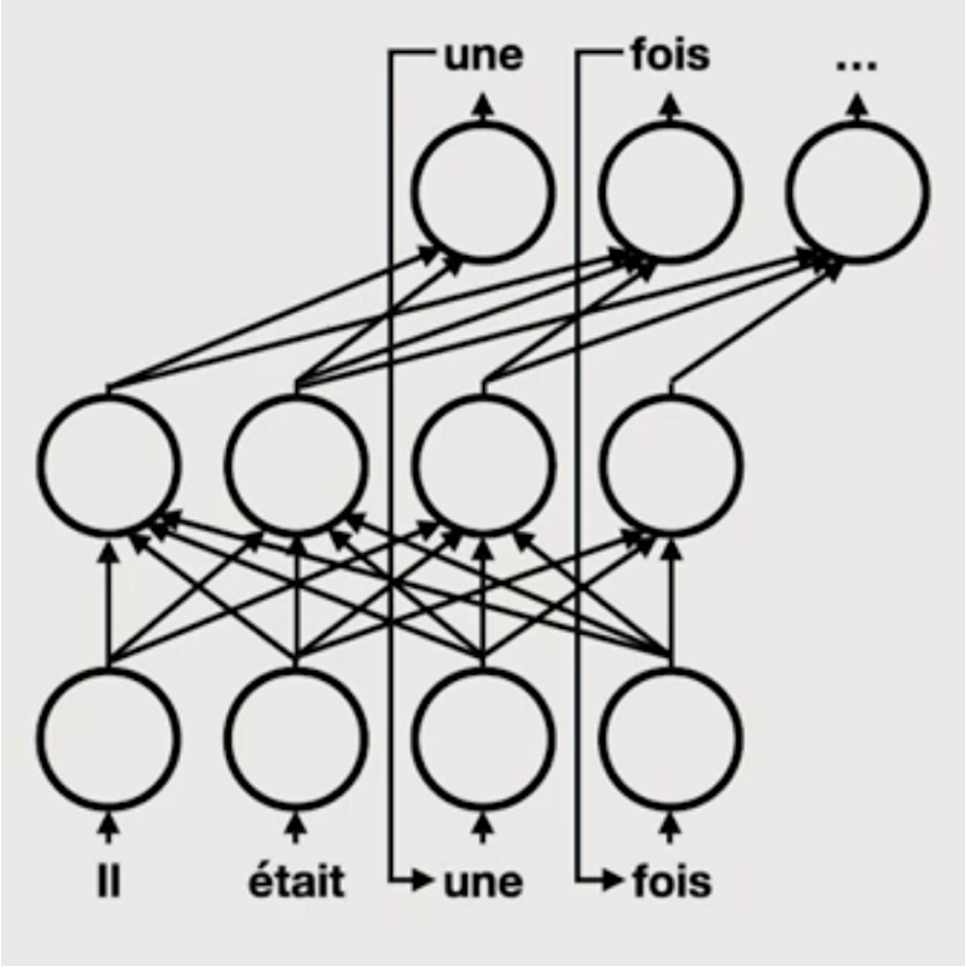
Taken from : https://www.jeremyjordan.me/intro-to-neural-networks/

Réseau multi-couches



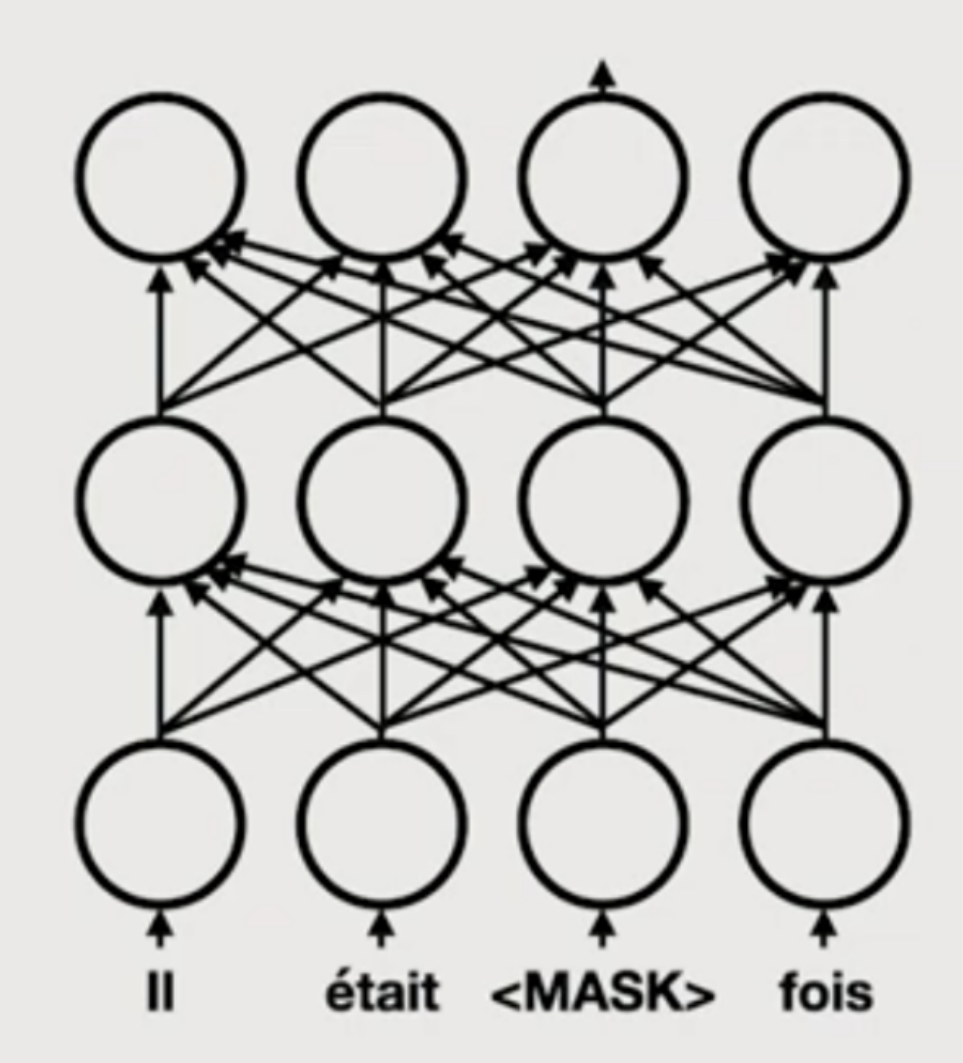
Taken from : https://cs231n.github.io/neural-networks-1/

Réseau causal



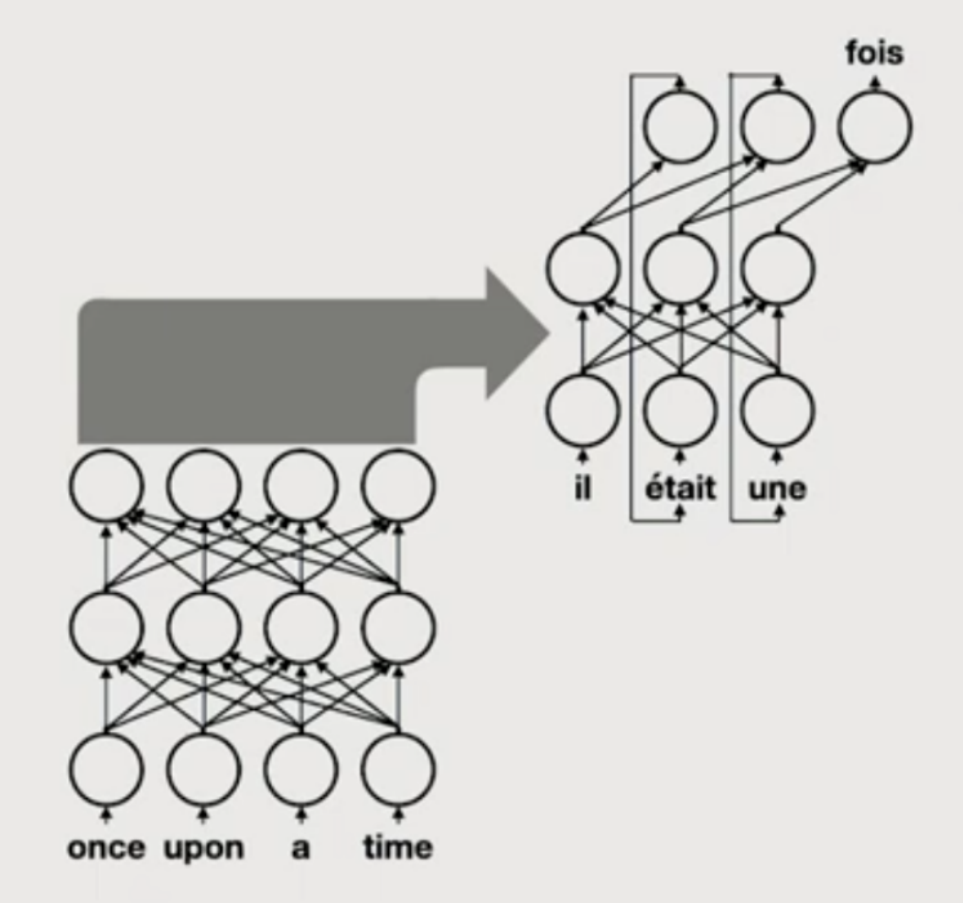
Taken from : B. Sagot

Réseau par masquage



Taken from : B. Sagot

Réseau conditionnel



Taken from : B. Sagot

Perplexité Généralisation : donner une probabilité élevée à des séquences de test jamais vues lors de l'apprentissage.

Plus la perplexité est basse, "meilleur" est le modèle.

Limité par le fait que le données de test doivent être similaires aux données d'apprentissage.

2.6 4e révolution de l'accès à l'information

Extrait de la leçon inaugurale de Benoît Sagot au Collège de France (11/2023) :

1. Écriture : stockage des informations de manière externe et pérenne. Outil d'accès à l'information ;
2. Imprimerie : externalisation et diffusion facilités ;
3. Web : numérisation massive, moteurs de recherche. Automatisation de l'identification des sources ;
4. IA : restitution des informations et capacité externe de raisonnement.

3 Grammaires formelles

3.1 Informatique et langages

Relation forte depuis l'origine de l'informatique en tant que science.

3.2 Langage formels, Hiérarchie de Chomsky

https://fr.wikipedia.org/wiki/HiÃľrarchie_de_Chomsky

3.3 Langages rationnels

Langages réguliers, expressions régulières.

3.4 Langages algébriques

Langages hors-contexte.

Langages de programmation.

3.5 Langages contextuels

Langages sensibles au contexte.

Langues naturelles se situent entre langages contextuels et algébriques, "langages légèrement sensibles au contexte".

3.6 Langages récursifs

Langages récursivement énumérables.

Programmes.

3.7 La machine de Turing

Objet mathématique abstrait composé :

- d'une bande infinie découpé en cases pouvant contenir un symbole ;
- d'une tête de lecture pouvant à chaque étape lire un symbole, écrire un symbole, puis se déplacer sur la bande d'une case à gauche ou à droite ;
- un registre fini d'états dans lesquels peut se trouver la machine ;
- une table d'action indiquant pour un état et un symbole l'action à effectuer.

Une machine de Turing déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

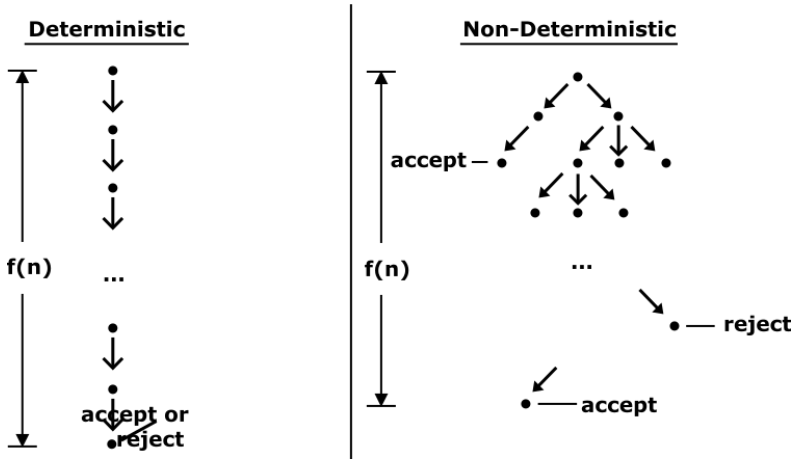
- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
- $\delta : (Q \setminus F) \times \Gamma \not\to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction partielle de *transition*. Si δ n'est pas définie sur l'état courant et le symbole courant, la machine s'arrête ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si elle s'arrête dans un état de F.

Exemple de partie de δ : $\delta(q_1, x) = (q_2, y, \leftarrow)$ indique que dans l'état q_1 quand x est lu sur la bande, on passe en état q_2, on écrit y et on se déplace à $\leftarrow$.

Une machine de Turing non déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;

— $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
— $\delta \subseteq (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ est la relation de *transition* ;
— $q_0 \in Q$ est l'état initial ;
— $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si *une branche* s'arrête dans un état de F.



3.8 Expressions régulières

— Expressions régulières par génération d'un automate fini (Ken Thompson).
— `grep`, `lex`, analyseur lexical
— https://regexcrossword.com
— Python : `import re`
— hyperscan

Exemple 1 : utilisation dans un IDE

```
#define MAX_URI_COUNTRY 3
#define MAX_URI_CORPUS 5
#define MAX_URI_NATURE 70
#define MAX_URI_YEAR 5
#define MAX_URI_MONTH 3
#define MAX_URI_DAY 3
#define MAX_URI_NUMBER 30
#define MAX_URI_VERSION 9
#define MAX_URI 256
```

```
MAX_(\w+)
$1_MAX
```

Exemple 2 : découpage d'un arrêt de cour d'appel d'Agen

```
intro_re = re.compile(
  r'^(?P<intro>.*?)(?=('
  r'<p>\s*A\s+rendu\s+l.arrêt\s+((réputé\s+)?
  r'contradictoire|par\s+défaut)'
  r'|<p>\s*EXPOS(É|E)\s*DU\s*LITIGE'
  r'|<p>A rendu réputé l.arrêt réputé contradictoire'
  r'))',
  re.UNICODE|re.DOTALL|re.MULTILINE|re.IGNORECASE)
decision_re = re.compile(
  r'(?P<decision>(<p>par\s*ces\s*motifs).*)$',
```

```
  re.U|re.DOTALL|re.MULTILINE|re.IGNORECASE)
```

Exemple 3 : numéros d'alinéas (Droit Quotidien)

```
alinea_number = (
  r"("
  r"\w\)(?=\s+)"
  r"|\d{1,2}°(\s+bis)?(?=\.?\s+)"
  r"|\d{1,2}(\s+bis)?(?=\.?\s+)"
  r"|[IVX]+(?=\.\s+)"
  r")"
)
```

3.9 Travaux pratiques sur les expressions régulières : Judilibre

https://github.com/gasilber/tp_nlp_regexps_mines

4 Modèle de langue bi-grammes par comptage

4.1 Implémentation via un notebook Jupyter

Notebook : https://mines.paris/nlp/notebooks/001_makemore_bigrams/makemore_bigrams_final.ipynb

Pour utiliser ce *notebook* en local sur votre machine, un première possibilité est de créer un environnement virtuel python et d'y installer les paquetages nécessaires au lancement de Jupyter Lab :

```
python3 -m venv .venv/notebooks
source .venv/notebooks/bin/activate
pip install -r requirements.txt  # Contenu du fichier plus bas
jupyter lab
```

Fichier `requirements.txt` à créer :

```
jupyterlab
numpy
matplotlib
graphviz
torch
```

Vous aurez également besoin du fichier `civil_mots.txt` que vous pouvez télécharger ici.

4.2 Points importants : TL ; DR du notebook

Représentation des `nb_chars` caractères sous la forme d'entiers (*tokens*).

Utilisation d'un tenseur PyTorch d'ordre 2 pour stocker le nombre d'occurences des bigrammes :

```
N = torch.zeros((nb_chars, nb_chars), dtype=torch.int32)
```

Chaque compte est un *logit* (*logistic unit*), c'est-à-dire une mesure "brute", que l'on va ensuite chercher à interpréter comme une probabilité, en ramenant ces logits à des valeurs entre 0 et 1, avec une somme égale à 1.

D'un pont de vue mathématique, un logit est le logarithme des "cotes" (ou *odds* en anglais) : si p est la probabilité qu'un événement se produise (entre 0 et 1), le logit est défini par la formule

$$\text{logit}(p) = \ln(\frac{p}{1-p})$$

où $\frac{p}{1-p}$ représente les cotes (le ratio entre la chance que quelque chose arrive et que quelque chose n'arrive pas).

Pour passer des logits à des probabilités, dans notre cas, on fait un "genre de" *softmax* :

```
P = N.float()  # Passage de int à float
P /= P.sum(1, keepdims=True)
```

La "vraie" fonction softmax étant définie comme :

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Génération de mots à partir du modèle :

```
g = torch.Generator().manual_seed(2147483647)

def generate_word(P, g):
    out = []  # Caractères du mot généré
    ix = 0  # Token du premier caractère '.'
    while True:  # On échantillone jusqu'à rencontrer un nouveau '.'
        p = P[ix]  # Vecteur des probabilités des bi-grammes commençant par ix
        # Tirage aléatoire d'un nouveau caractère par rapport à la distribution 'p'
        ix = torch.multinomial(p, num_samples=1, replacement=True, generator=g).item()
        if ix == 0:  # Si c'est un '.' on s'arrête
            break
        out.append(itoc[ix])  # Ajout du caractère tiré
    return ''.join(out)
```

Voir torch.Generator() et torch.multinomial().

Les mots générés sont par exemple :

```
éssanée
mexcororér
monts
ex
moût
```

4.3 Fonction de perte utilisant la "log-vraissemblance" (*log-likelihood*)

Fonction de perte (*loss*) permettant de mesurer la "qualité" du modèle à paramètres constants. On cherche à maximiser la vraissemblance (*likelihood*) des données, considérant les paramètres du modèle. Il est équivalent et plus commode de maximiser le logarithme naturel de cette vraisemblance (log est strictement croissant), ce qui revient à minimiser l'opposée de cette log-vraissemblance. En pratique, on va cherche à minimiser la moyenne de l'opposée de la log-vraissemblance.

```
def loss_score(words:list, P:torch.tensor):
    log_likelihood = 0.0
    n = 0
    for w in words:
        chs = ['.'] + list(w) + ['.']
        for ch1, ch2 in zip(chs, chs[1:]):
            ix1 = ctoi[ch1]
            ix2 = ctoi[ch2]
            prob = P[ix1, ix2]
            logprob = torch.log(prob)
            log_likelihood += logprob
            n += 1
```

```
    nll = -log_likelihood
    score = nll / n  # Moyenne
    return score
```

Le modèle précédent donne un *score* (perte) de 2,34, qui représente la qualité de notre modèle par comptage.

Note : si tous les caractères étaient équiprobables,

4.4 Lissage

Notre modèle se comporte de manière stricte lorsqu'il rencontre une combinaison de caractères qu'il ne connaît pas. Par exemple, si on cherche le score d'un mot comme `espérancejq`, où la combinaison `jq` a une probabilité 0, ce score est ∞, car selon le code précédent de la fonction de perte, on utilise la fonction log avec une valeur nulle, qui en mode flottant en python renvoie l'infini.

Habituellement, on fait donc du lissage de modèles (*smoothing*), en ajoutant par exemple 1 à tous les comptage avant de calculer les probabilités (noter le `N+1` ci-dessous) :

```
P = (N+1).float()
P /= P.sum(1, keepdims=True)
```

5 Rétropropagation du gradient

Notebooks :

- https://mines.paris/nlp/notebooks/002_micrograd/micrograd_1.ipynb
- https://mines.paris/nlp/notebooks/002_micrograd/micrograd_2.ipynb

Pour installer ce notebook, vous pouvez suivre les instructions du chapitre précédent.

6 Modèle de langue bi-grammes avec un réseau de neurones

Notebook : https://mines.paris/nlp/notebooks/003_makemore_neural/makemore_neural_final.ipynb

6.1 Travaux pratiques sur les tri-grammes

Reprendre l'exemple du modèle bi-grammes, par comptage et neuronal, pour passer à des tri-grammes, c'est-à-dire des suites de trois caractères.

À rendre : le notebook résultat ou le code python correspondant (par email).

7 Perceptron Multi-Couches

L'article de 2023 de Bengio *et al.*, "A Neural Probabilistic Language Model" [1], est l'un des premiers à proposer une solution efficace de type MLP (Multi-Layer Perceptron).

Idée clef : associer à chaque mot (ou caractère dans notre cas) un vecteur de caractéristiques (embedding) dans un espace de dimension réduite (par ex. 30 dimensions, ou 2 dimensions dans notre cas). Avantage : les mots ayant un sens similaire se retrouveront proches dans cet espace, permettant au modèle de généraliser à des séquences jamais vues auparavant.

8 Initialisations et normalisations

9 Réseau de type wavenet

10 Transformers

Il y a trois architectures de Transformer principales : encodeur-décodeur [7], utilisé typiquement pour de la traduction, encodeur seul [4], utilisé typiquement pour la classification, et décodeur seul [5], utilisé typiquement dans les IA génératives.

Nous allons principalement détailler le modèle décodeur seul, qui est l'approche la plus utilisée pour entraîner des modèles de langage auto-régressifs.

Notebook utilisé pour le cours : https://mines.paris/nlp/notebooks/007_gpt/gpt.ipynb

Voir également le notebook d'A. Burkov [2] où il construit également un décodeur de Transformeur : https://github.com/aburkov/theLMbook/blob/main/news_decoder_language_model.ipynb.

10.1 Mécanisme d'attention

10.1.1 Idée générale

L'introduction explicite des matrices $\boldsymbol{Q}$ (*query*), $\boldsymbol{K}$ (*key*) et $\boldsymbol{V}$ (*value*) a été popularisée par l'article fondateur "*Attention Is All You Need*" [7], qui a présenté l'architecture *Transformer*. Les termes *query*, *key* et *value* proviennent directement des systèmes de bases de données et de la recherche d'information, ici appliqués aux *tokens*.

L'attention elle-même a été introduite aux alentours de 2014 [3], par Dzmitry Bahdanau. Il s'intéressait avec Yoshua Bengio un challenge fondamental en traduction automatique [2, p.122] : apprendre à un RNN de se concentrer sur les morceaux les plus pertinents d'une phrase lors du processus de traduction. Ce mécanisme que Y. Bengion appela "*attention*", est devenu une pierre angulaire des réseaux de neurones modernes.

Ainsi, dans le mécanisme d'attention, chaque token va être associé à trois vecteurs, dans un espace à d_k dimensions :

— $\boldsymbol{q}$: ce que le token **recherche** ;
— $\boldsymbol{k}$: ce que le token **répond** ;
— $\boldsymbol{v}$: ce que le token **contient**.

L'intuition fondamentale est de dissocier ces trois sujets : la manière dont on recherche une information ($\boldsymbol{q}$) et la manière dont cette information est indexée ($\boldsymbol{k}$) ne doivent pas nécessairement être identiques à l'information elle-même ($\boldsymbol{v}$).

Les i vecteurs associés aux tokens présents dans la séquence d'entrée représentée par la matrice $\boldsymbol{X}$ seront associés à leurs vecteurs $\boldsymbol{q}_i$, $\boldsymbol{k}_i$ et $\boldsymbol{v}_i$ aggrégés dans les matrices $\boldsymbol{Q}$, $\boldsymbol{K}$ et $\boldsymbol{V}$.

Avant les Transformers, notamment dans les modèles Seq2Seq avec attention [3], l'attention était calculée directement entre les états cachés de l'encodeur et du décodeur.

L'état caché devait donc tout faire, représenter le sens du mot, servir de clé pour être comparé, et servir de valeur pour être additionné, limitant ainsi l'expressivité du modèle : un vecteur devait représenter simultanément "ce que je suis", "comment je me connecte aux autres" et "ce que j'apporte comme information".

L'introduction des projections apprises par le réseau $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$ et $\boldsymbol{W}^V$ a permis de découpler ces rôles :

— le rôle de $\boldsymbol{Q}$ (les questions) : le modèle apprend à transformer les tokens actuels en des questions. Par exemple, si le token actuel représente le mot "le", son vecteur $\boldsymbol{q}$ pourrait chercher (via l'attention) un nom commun qui suit ;

— le rôle de $\boldsymbol{K}$ (les réponses) : le modèle apprend à transformer chaque token en une "étiquette" ou une "clé". Le token associé au nom "chat" aura un vecteur $\boldsymbol{k}$ qui répondra fortement (similarité cosinus) à la question posée par le $\boldsymbol{q}$ de "le" ;
— le rôle de $\boldsymbol{V}$ (les contenus) : une fois que l'attention est établie (affinité entre $\boldsymbol{q}$ et $\boldsymbol{k}$), on ne veut pas forcément transmettre l'information grammaticale utilisée pour la correspondance. On veut transmettre le sens sémantique du mot. C'est le rôle de $\boldsymbol{V}$.

10.1.2 La formule de l'attention (*Scaled Dot-Product*)

Soit une séquence d'entrée $\boldsymbol{X} \in \mathbb{R}^{n \times d_k}$. Les projections sont définies par :

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

et l'attention est calculée ainsi, avec $\boldsymbol{M}$ un masque d'attention permettant de ne pas tenir compte de certaines valeurs positionnées dans le masque à $-\infty$ (les autres étant positionnées à 0) :

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}} + \boldsymbol{M}\right)\boldsymbol{V}$$

Par exemple, dans un modèle génératif causal, le masque assure que les tokens ne peuvent prêter attentions qu'aux tokens les précédants. D'autres utilisations de ce masque sont possibles, par exemple pour implémenter du "*Continuous batching*".

10.1.3 Utilité des projections

Asymétrie. Si nous n'utilisions pas de matrices de projection ($\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{X}$), la formule de similarité deviendrait $\boldsymbol{X}\boldsymbol{X}^T$: la matrice d'attention serait symétrique. Or, le langage est intrinsèquement **asymétrique** : dans la phrase "Le chat mange", le verbe "mange" doit porter une forte attention à "chat" pour s'accorder, mais l'inverse n'est pas vrai.

En utilisant $\boldsymbol{W}^Q$ et $\boldsymbol{W}^K$, nous calculons $(\boldsymbol{X}\boldsymbol{W}^Q)(\boldsymbol{X}\boldsymbol{W}^K)^T$ et cela brise la symétrie : même si les entrées sont fixes, la relation devient dirigée et apprenable.

Par ailleurs, sans matrices de projection, un vecteur x_i devrait être géométriquement proche des vecteurs avec lesquels il interagit : pour que le pronom "Il" s'aligne avec le nom "Pierre" dans une phrase, leurs vecteurs devraient être proches (produit scalaire élevé). Pourtant, sémantiquement, un pronom et un nom propre sont des entités différentes et ne devraient donc pas occuper la même place dans l'espace latent.

Les matrices résolvent ce conflit par projection :

— $\boldsymbol{W}^K$ projette "Pierre" dans un sous-espace "Antécédent Masculin" ;
— $\boldsymbol{W}^Q$ projette "Il" dans ce même sous-espace en tant que "Recherche d'Antécédent".

Cela permet un "match" fort sans corrompre la représentation sémantique originale des mots.

Séparation de l'alignement et du contenu. La sortie de l'attention est une somme pondérée $\boldsymbol{z}_i = \sum_j \alpha_{ij}\boldsymbol{v}_j$. Si $\boldsymbol{V} = \boldsymbol{X}$, la sortie ne serait qu'une moyenne des entrées.

La matrice $\boldsymbol{W}^V$ agit comme un sélecteur de caractéristiques. Elle permet transmettre uniquement l'information pertinente pour la couche suivante (par exemple, conserver l'information grammaticale mais ignorer le style), indépendamment de l'information utilisée pour l'affinité ($\boldsymbol{Q}$ et $\boldsymbol{K}$).

Expressivité et *Multi-Head Attention*. Enfin, l'utilisation de h têtes d'attention (*Multi-Head*) implique h ensembles de matrices $\{W_i^Q, W_i^K, W_i^V\}$, permettant de projeter l'entrée $\boldsymbol{X}$ dans plusieurs sous-espaces différents simultanément. Une tête peut se spécialiser sur la grammaire, une autre sur les références temporelles, et une autre sur les liens causaux. Sans ces projections spécifiques, le modèle ne pourrait calculer qu'une unique "moyenne" de toutes les relations possibles.

Synthèse. L'introduction des matrices $\boldsymbol{Q}$, $\boldsymbol{K}$ et $\boldsymbol{V}$ transforme une simple opération de moyenne basée sur la similarité en un mécanisme de **routage d'information flexible**. Elles permettent :

1. de créer des relations asymétriques ($\boldsymbol{Q}, \boldsymbol{K}$) ;
2. de séparer l'alignement du contenu ($\boldsymbol{V}$) ;
3. d'explorer plusieurs sous-espaces sémantiques simultanément (Multi-Head).

10.2 Rotary Position Embedding (RoPE)

11 Fine-tuning

https://github.com/aburkov/theLMbook/blob/main/emotion_classifier_LR.ipynb

https://github.com/aburkov/theLMbook/blob/main/emotion_GPT2_as_text_generator.ipynb

Références

[1] Yoshua Bengio et al. "A neural probabilistic language model". In : *The Journal of Machine Learning Research* (1er mars 2003). URL : https://dl.acm.org/doi/10.5555/944919.944966 (cf. p. 14).

[2] Andriy Burkov. *The Hundred-Page Language Models Book*. 2025. URL : https://www.thelmbook.com/ (cf. p. 15).

[3] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2 sept. 2014. arXiv : 1406.1078[cs,stat]. URL : http://arxiv.org/abs/1406.1078 (cf. p. 15).

[4] Jacob Devlin et al. "BERT : Pre-training of deep bidirectional transformers for language understanding". In : *CoRR* abs/1810.04805 (2018). URL : http://arxiv.org/abs/1810.04805 (cf. p. 15).

[5] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. 2019. URL : https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (cf. p. 15).

[6] Alan Mathison Turing. "Computing machinery and intelligence". In : *Mind ; a quarterly review of psychology and philosophy* LIX.236 (oct. 1950), p. 433-460. URL : https://doi.org/10.1093/mind/LIX.236.433 (cf. p. 2).

[7] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv : 1706.03762[cs]. URL : http://arxiv.org/abs/1706.03762 (cf. p. 15).