# Machine Learning

EMSI - Université Côte d'Azur
Richard Grin
Version 1.34.2 - 1/10/25

1

## Objectif du cours

- Ce cours « Agents conversationnels en Java avec LangChain4j » fait partie du parcours MIAGE « Intelligence Artificielle Appliquée » (IA2)
- Il est essentiellement pratique en montrant comment tirer profit de LMs dans les applications d'entreprise, en utilisant l'API de ces modèles
- LM : *Language Model* ; modèle de langage utilisé, par exemple, par ChatGPT et Gemini
  LLM : *Large Language Model*



2

## Bonus

- Pour la réactivité : réponse aux emails en respectant les formats demandés, installation des logiciels, TPs terminés,…
- Pour les réponses aux questions pendant le cours
- Pour les questions intéressantes
- Retenez votre numéro dans la liste ; vous me donnerez ce numéro si vous avez un bonus, ou dans vos emails
- Malus pour retards, manque de travail manifeste, pas pour mauvaises réponses ou questions



3

## Examen

- Durée 2 heures, sans documents
- Les bonus sont ajoutés à la note de l'examen pour avoir la note finale
- Vous choisissez la date, avec l'administration de l'EMSI ; au moins 15 jours après la fin du cours, pour avoir le temps de finir les TPs, car des questions pourront porter sur les TPs
- Le délégué me tient au courant assez rapidement



4

## TPs

- Ils **indispensables** pour assimiler le cours
- Bonus pour les TPs, seulement si projet GitHub avec tous les commits
- Vous pourrez terminer les TPs après mon départ et avant l'examen
- Il vous est demandé un travail personnel ; vous pouvez demander des explications (pas le code déjà écrit) à votre entourage ou à une IA, mais vous devez écrire le code seul, en suivant ce qui est demandé dans les énoncés des TPs



5

## Comment demander de l'aide

- Lisez attentivement cette page : http://richard.grin.free.fr/emsi/casablanca-ia/tp/demandeaide.html
- En résumé :
  - N'envoyez pas de copie d'écran, sauf exception
  - Le minimum d'information à fournir :
    - environnement d'exécution (version de l'OS et des logiciels,…)
    - étapes qui ont conduit au problème
    - message d'erreur (et logs)



6

## Supports du cours

- Premier support rappelle quelques généralités et concepts importants du machine learning
- Deuxième support sur les embeddings et l'utilisation des LMs
- Troisième support sur LangChain4j, le framework standard de facto pour utiliser les LMs avec du code Java, utilisé dans les TPs
- Dernier support sur le RAG qui permet d'améliorer les réponses des LMs



7

## Plan de ce support





8

## Intelligence artificielle



9

## Définitions

- IA : ensemble des théories et techniques visant à réaliser des machines capables de simuler l'intelligence humaine (Wikipedia)
- Le but est d'automatiser, de rendre plus rapides et plus efficaces des tâches qui demandent de l'intelligence
- Intelligence : capacité à comprendre, apprendre, s'adapter et résoudre des problèmes dans un environnement donné



10

## Types d'IA

- IA « classique » : s'appuie sur des règles et des connaissances fournies par des experts ; fournit des systèmes experts
- Machine learning (ML) : développe des algorithmes et des modèles permettant à des ordinateurs
  - de faire des prédictions
  - en s'appuyant sur des données d'entrainement fournies lors d'une phase d'apprentissage
  - sans codage manuel des algorithmes



11

## Machine learning



12

## Présentation

- 2 phases bien distinctes :
  - Durant la **phase d'apprentissage** le logiciel apprend à partir d'exemples (*dataset*) qu'on lui fournit ; cette phase fournit un **modèle**
  - Durant l'**inférence**, le modèle est utilisé pour prévoir des résultats, comme un logiciel sans IA
- L'apprentissage est souvent très lourd et coûteux mais ne se fait qu'une seule fois ; nécessite des machines très puissantes, avec GPU ou NPU (on peut en louer sur le cloud si utilisation peu fréquente)



13

## Modèle

- Logiciel qui permet de faire des prédictions ou de prendre des décisions
- Représentation mathématique $y = f(x_1, ..., x_n)$ ; trouver un résultat $y$ à partir des données d'entrée $x_i$
- Pour le type régression, $y$ est numérique (valeur simple, vecteur,…) ; pour le type catégoriel, $y$ est une catégorie (classification) ; pour le type génératif, $y$ peut être du texte, une image, une vidéo, du son,…



14

## Exemples d'apprentissage

- On entraine le modèle en lui donnant des exemples de ce que l'on veut
- Chaque exemple est composé
  - de valeurs $x_1, ..., x_n$ en entrée
  - d'une réponse $y$ attendue pour ces valeurs en entrée
- Pour chaque exemple, le fonctionnement du modèle est ajusté pour que le $y$ calculé soit le plus proche possible de la réponse attendue
- Pour éviter le bruit (chaque exemple tire les paramètres dans « sa direction »), on regroupe souvent les exemples dans des *batchs* (groupe de quelques dizaines à quelques centaines d'exemples)



15

## Types de machine learning

- Dans le ML « classique », le *data scientist* choisit un type de modèle à utiliser (modèle linéaire, polynomial, arbre de décision, clustering, par exemple)
- Le deep learning (DL, apprentissage profond) a pour modèle un réseau de neurones ; à privilégier si les données sont complexes, non structurées (texte, image, audio) et volumineuses



16

## Data scientist

- Expert en science des données
- Spécialiste capable d'extraire de la valeur à partir de données brutes
- En ML, ses tâches principales sont
  - choix et préparation des données d'entrainement
  - choix du type de modèle
  - évaluation du modèle



17

## IA générative (*GenAI*)

- Branche du DL qui génère des contenus nouveaux
- Les **LLM**s (*Large Language Models*) sont les logiciels d'IA générative les plus utilisés (ChatGPT, Gemini,…) ; ils permettent de traiter les langages naturels (français, anglais, …) : chat avec les utilisateurs, traduction de texte, aide au codage, …
- Des logiciels d'IA générative peuvent générer du texte mais aussi du code, des images, de la musique, des vidéos, … ; on parlera alors de LLMs **multimodaux** ou de **modèles de base**



18

## Cas d'utilisation

- ML « classique », souvent avec des données structurées :
  - Prédiction de défauts de paiement (finance)
  - Classification de patients à risque (santé)
  - Décider si un email est un spam
  - Optimisation des stocks
- DL, souvent sur des données complexes non structurées (textes, images, sons) :
  - Traitement du langage naturel (traduction, chatbot)
  - Vision par ordinateur (reconnaissance faciale, détection d'objets, diagnostic médical par imagerie)
  - Voitures autonomes, robots « intelligents »



19

## Exemples de types de modèles

- Linéaire - fonction linéaire entre les variables d'entrée et de sortie (exemple : régression linéaire)
- Arbre de décision - utilise une structure d'arbre pour prendre des décisions
- Clustering - regroupe des données en clusters (groupes) selon leur similarité, sans étiquettes prédéfinies (exemple : k-means clustering)
- Réseaux de neurones - utilise des neurones artificiels



20



## Exemple 1 de modèle

- Modèle qui prédit si un client est susceptible d'acheter un certain produit
- Un client est décrit par des caractéristiques qui correspondent aux données fournies pendant l'apprentissage : âge, groupe social ou géographique, achats déjà effectués, etc.
- Le modèle est entrainé avec de nombreux exemples **réels** de clients, avec leurs caractéristiques, en précisant si le client a acheté le produit
- Après l'entrainement, le modèle doit être capable de prédire (et avec quelle probabilité) si un client quelconque est susceptible d'acheter le produit



21



## Exemple 2 de modèle

- Modèle qui calcule le prix d'un appartement à afficher, compte tenu de son emplacement, de sa superficie, de son étage, de son environnement, … pour qu'il trouve rapidement un acquéreur
- Pendant l'entrainement, on fournit au modèle les caractéristiques de très nombreux appartements qui ont été vendus, le prix de vente et le temps qu'il a fallu pour trouver un acquéreur



22



## Exemple 3 de modèle

- Modèle qui indique si une photo contient des installations militaires
- Pendant l'entrainement, on fournit au modèle les caractéristiques de très nombreuses photos que des hommes ont classées « militaire » ou non



23



## Exemple 4 de modèle

- Modèle qui génère une image à partir d'une description textuelle
- Pendant l'entrainement, on fournit au modèle des couples texte - image ; il existe des BD de tels couples



24

## Paramètres d'un modèle

- La plupart des types de modèle ont des paramètres définis à l'avance p1, p2, … qui déterminent le comportement du modèle
- Par exemple, si on choisit un modèle linéaire, le modèle est représenté par une fonction affine qui a 2 paramètres **a** et **b** : f(x) = **a**x + **b**
- Les valeurs de ces paramètres sont déterminées automatiquement durant l'apprentissage pour minimiser les erreurs de prédiction sur les données d'apprentissage



25

## Paramètres et hyperparamètres

- Les **paramètres** d'un modèle sont *internes* au modèle ; leurs valeurs sont *automatiquement* ajustées pendant l'apprentissage, comme les valeurs a et b d'un modèle linéaire
- Ne pas confondre avec les **hyperparamètres** qui sont des valeurs *choisies* pour paramétrer le modèle
  - pendant l'apprentissage du modèle : taille des pas pendant la descente de gradient, nombre des couches cachées des réseaux de neurones, …
  - pendant l'inférence : température, taille maximale de la réponse,…



26

## Eléments pour apprentissage

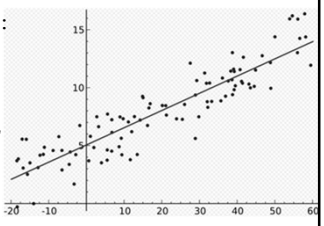
- Nombreux types d'apprentissage mais ils utilisent tous plus ou moins les mêmes éléments :
  - **Exemples d'apprentissage** ; chaque exemple est composé d'une valeur qu'on donne en entrée du modèle, et du résultat attendu pour cette entrée
  - **Modèle** qui apprend à partir de ces exemples
  - **Fonction de perte** qui mesure l'erreur faite par le modèle lorsqu'il est testé sur un batch d'apprentissage
  - **Algorithme d'optimisation** qui modifie le modèle pour réduire la fonction de perte sur les exemples d'apprentissage



27

## Exemple régression linéaire (1/2)

- **Données d'apprentissage** : points qui correspondent aux données relevées
- **Modèle** : fonction affine f(x) = ax + b représentée par une droite de régression



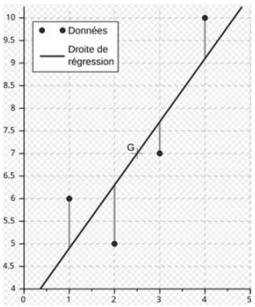

Dessin Wikipedia



28

## Exemple régression linéaire (2/2)

- **Fonction de perte** : « somme des distances » des points à la droite de régression
- **Algorithme d'optimisation** : minimisation de la somme, par exemple avec une descente de gradient






29

## Etapes phase d'apprentissage

1. Préparation des données d'apprentissage
2. Fixer les hyperparamètres d'apprentissage (par exemple, nombre de couches de neurones ou taux d'apprentissage)
3. Utiliser les exemples d'apprentissage pour trouver les paramètres internes du modèle
4. Validation du modèle sur les exemples de validation ; on peut être amené à changer les hyperparamètres et en ce cas il faut recommencer l'étape 3
5. Test du modèle avec les exemples de tests



30

## Préparer données apprentissage

- Très important
- Les données en entrée doivent souvent être préparées pour supprimer le bruit et les données erronées ; par exemple suppression de balise HTML, correction fautes d'orthographe, espaces superflus, …
- Eliminer les biais
- Gestion des données sensibles (enlever les mots de passe, les clés secrètes)
- Ajout éventuel de contexte





## Diviser données apprentissage

- En 3 groupes :
  - Le plus nombreux pour calculer les paramètres internes du modèle (par exemple 80 % des exemples)
  - Un pour trouver les meilleurs hyperparamètres d'apprentissage (10 %)
  - Le dernier pour tester l'efficacité du modèle final (10 %)





## Apprentissage (1/2)

- But : ajuster les paramètres pour le modèle
- On commence par donner des valeurs (aléatoires, si pas d'informations spéciales) aux paramètres du modèle
- Pour chaque exemple (ou groupe d'exemples, *batch*) d'apprentissage
  - La fonction de perte dépend des paramètres du modèle (calcul de la valeur de sortie du modèle) et de l'exemple (valeur en entrée et attendue en sortie)
  - L'algorithme d'optimisation modifie les paramètres du modèle pour minimiser la perte





## Apprentissage (2/2)

- Avec les nouveaux paramètres mis à jour, on refait le même processus avec l'exemple d'apprentissage suivant, et ainsi de suite, jusqu'au dernier exemple d'apprentissage (fin d'une époque)
- On peut alors commencer une nouvelle époque avec des exemples différents, ou identiques mais pas dans le même ordre
- L'apprentissage s'arrête quand la perte cesse de diminuer, pour éviter le surapprentissage (des métriques surveillent le processus)





## Surapprentissage

- Il faut savoir s'arrêter à temps dans l'apprentissage
- Si on s'arrête trop tard, on va faire du sur-apprentissage (overfitting) : le modèle obtenu sera surentrainé sur les données d'entrainement mais ne donnera pas des bons résultats avec d'autres données


Source images : https://www.youtube.com/watch?v=kwgR-Qu56MA





## Algorithme descente de gradient

- Algorithme d'optimisation pour trouver des paramètres du modèle qui minimisent la fonction de perte
- Rappel : Le gradient en un point d'une fonction à plusieurs variables est le vecteur des dérivées partielles par rapport à chacune des variables
  $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
  Il pointe vers l'endroit où la fonction augmente le plus vite
- Si on veut minimiser la valeur de la fonction en partant d'un point $(x_1, \ldots, x_n)$, on modifie les $x_i$ en allant dans la direction opposée à la direction du gradient
- Voir la fonction de perte comme une hypersurface donne une intuition de la descente de gradient

## Fonction de perte représentée par une hypersurface

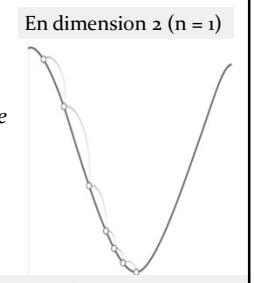
- La fonction de perte L dépend des n paramètres du modèle et elle fournit une seule valeur (l'erreur)
- Elle peut donc être représentée par une hypersurface de dimension n dans un espace de dimension n + 1
- Par exemple,
  - si n = 1, une hypersurface est une courbe dans le plan ; fonction de perte de type `y = L(p1)`
  - si n = 2, une hypersurface est une surface dans l'espace ; fonction de perte de type `y = L(p1, p2)`



37

## En dimension 2 (n = 1 paramètre)

- L'algorithme de la descente de gradient revient à se déplacer dans le plan (donc à modifier l'unique paramètre) *en prenant la direction de la plus grande pente descendante* pour minimiser la valeur de la fonction de perte






38

## En dimension 3 (n = 2 paramètres)

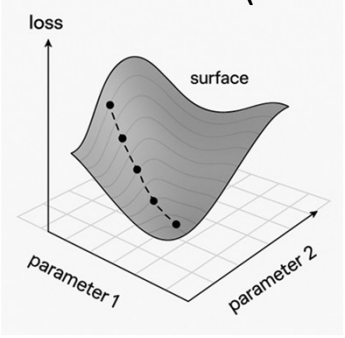



Image générée par IA



39

## Taux d'apprentissage

- La longueur du déplacement en direction de la plus grande pente est un hyperparamètre de l'algorithme d'apprentissage, qui s'appelle le *learning rate*
- Si le pas est trop petit, l'algorithme se rapproche trop lentement du minimum ; s'il est trop grand, on risque de ne pas se rapprocher du minimum (oscillation autour du minimum)



40

## Test du modèle

- Il faut garder une partie (par exemple 10 %) des exemples d'apprentissage pour tester le modèle
- Ces exemples doivent être différents des exemples d'apprentissage pour détecter le surapprentissage



41

## Types d'apprentissages

- Supervisé (supervised learning)
- Non supervisé (unsupervised learning)
- Auto-supervisé (self-supervised learning)
- Par renforcement (reinforcement learning)



42

## Apprentissage supervisé

- Un être humain fournit des exemples de ce qu'on attend : des données structurées et étiquetées, avec des entrées (caractéristiques, *features*) et des sorties (variables cibles)
- Par exemple
  - Classer des photos de chiens et de chats en partant d'exemples d'images étiquetées « chien » ou « chat »
  - Estimer le prix d'un appartement (variable cible) à partir de ses caractéristiques (étage, emplacement, superficie,…) en partant des caractéristiques et prix d'appartements récemment vendus



43

## Apprentissage non supervisé

- Le logiciel doit trouver des patterns ou des relations entre des données, sans recevoir d'aide
- Les données d'entrainement ne sont pas étiquetées
- L'objectif peut être
  - d'extraire des classes d'individus présentant des caractéristiques communes (clustering) ; les définitions des classes ne sont pas données a priori
  - de trouver des associations entre les données ; par exemple indiquer que les clients qui achètent du pain achètent aussi souvent du beurre
- Moins performant que l'apprentissage supervisé mais intéressant si l'étiquetage est complexe ou coûteux



44

## Apprentissage auto-supervisé

- Intermédiaire entre apprentissage supervisé et non supervisé
- Le modèle apprend à partir d'échantillons de données non annotées ; des « étiquettes » sont générées automatiquement à partir des données d'entrainement
- Par exemple, l'entrainement d'une IA textuelle générative peut se faire en tronquant des phrases récupérées dans un livre ou sur Internet ; l'IA doit retrouver la partie cachée



45

## Apprentissage par renforcement

- Le logiciel apprend en interagissant avec un environnement
- Il reçoit un feedback (retour d'information) en recevant des récompenses ou des pénalités en fonction des actions qu'il entreprend dans l'environnement
- Souvent utilisé quand les données d'entrainement sont rares ; l'IA apprend par essais et erreurs



46

## Apprentissage par renforcement avec rétroaction humaine

- RLHF : Reinforcement Learning from Human Feedback
- Utilisé pour affiner les modèles, en particulier les LLMs
- Exemples d'interactions :
  - Des testeurs humains donnent leur avis sur les réponses des modèles
  - Plusieurs réponses peuvent être proposées aux testeurs par le modèle, en demandant la meilleure réponse
  - Demander aux testeurs de sélectionner avec la souris des objets dans une image



47

## Distillation des connaissances

- Apparue avec les LLMs
- Technique d'apprentissage automatique qui transfère les connaissances d'un grand modèle, le modèle « enseignant » à un modèle « élève » plus petit
- On peut ainsi avoir un modèle plus petit, plus rapide, moins gourmand en ressources tout en gardant des connaissances proches du modèle enseignant
- Il peut y avoir des inconvénients : perte de connaissance, ne réduit pas forcément le coût d'entrainement, peut être difficile à optimiser



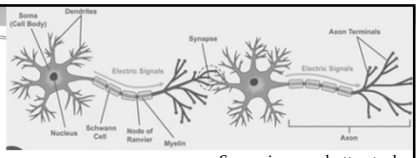
48

# Réseaux de neurones



49

## Cerveau humain



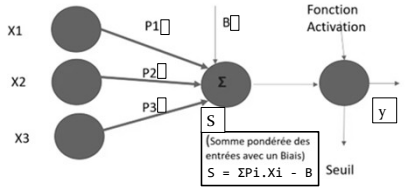

Source image : shutterstock

- 86 milliards de neurones
- Par neurone, environ 2000 dendrites (points d'entrée) ; le plus souvent un seul axone (point de sortie) mais avec plusieurs ramifications
- Chaque synapse (connexion axone-dendrite) peut être utilisée plusieurs centaines de fois par seconde
- Très efficace en énergie : consommation de 20 watts (quelques kwatts pour une IA)
- Apprendre c'est créer, supprimer des synapses, modifier leur sensibilité



50

## Au début : le perceptron




Frank Rosenblatt, 1957

- Un perceptron a des paramètres `Pi` (**poids**) et `B` (seuil ou **biais**)
- Il reçoit n valeurs `Xi` en entrée et calcule la somme pondérée `S` de ces valeurs ; la fonction d'activation retourne la valeur finale `1` si `S - B` ≥ `0`, et `0` sinon



51

## Neurones artificiels

- La fonction d'activation du perceptron est trop simple et elle n'est pas continue, donc pas dérivable
- Les neurones artificiels « modernes » sont des évolutions du perceptron avec, en particulier, des fonctions d'activation dérivables qui vont permettre d'utiliser la descente de gradient



52

## Fonction d'activation

- Perceptron : la sortie du perceptron est y = H(ΣPi1.Xi - B), avec H(z) = 1 si z ≥ 0, ou 0 sinon
- Neurone artificiel : on remplace H par une fonction dérivable
- Par exemple, avec la fonction d'activation sigmoïde (d'autres choix sont possibles), la sortie du neurone est donnée par ($w_i$ sont les poids, b est le biais)

$$y = \frac{1}{1 + e^{-(\sum w_i x_i + b)}}$$



53

## Types de fonctions d'activation

1. **Sigmoïde :** $\sigma(x) = 1 / (1+e^{-x})$ - Produit des sorties dans la plage de 0 à 1 ; utilisée dans les couches de sortie pour les problèmes de classification binaire
2. **Tangente hyperbolique (tanh) :** $\tanh(x) = (e^{-x} - e^{-x}) / (e^{-x} + e^{-x})$ - Produit des sorties dans la plage de -1 à 1, permet de gérer les valeurs négatives
3. **ReLU (Rectified Linear Unit) :** $\mathrm{ReLU}(x) = \max(0, x)$ - Renvoie simplement zéro pour les valeurs négatives et laisse les valeurs positives inchangées. Une des fonctions d'activation les plus populaires pour les couches cachées
4. **Leaky ReLU :** Leaky $\mathrm{ReLU}(x) = \max(\alpha x, x)$ - Variation de ReLU qui permet un faible taux de fuite ($\alpha$ est une petite valeur constante pour les entrées négatives) pour éviter certains problèmes liés à ReLU
5. **Softmax :** Souvent utilisée dans la couche de sortie pour les problèmes de classification multi-classes - convertit un vecteur de nombres réels en une distribution de probabilité



54

## Réseaux de neurones artificiels

- Les neurones artificiels sont regroupés en réseaux
- Ces réseaux sont structurés en couches de neurones
- Les valeurs calculées par les neurones d'une couche sont envoyées en entrée des neurones de la couche suivante



55

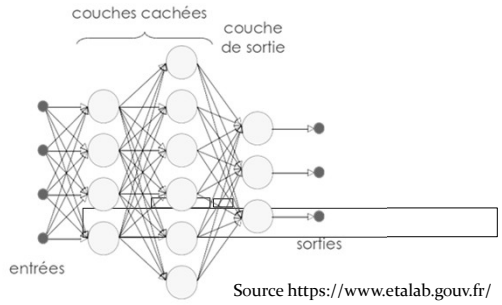
## Couches réseau de neurones

- La couche d'entrée est composée de neurones qui reçoivent des valeurs de l'extérieur du réseau
- La couche de sortie fournit à l'extérieur un résultat calculé par le réseau de neurones
- Entre ces 2 couches, les couches intermédiaires, dites cachées, participent au résultat calculé par le réseau
- Souvent, les neurones d'une couche sont connectés à tous les neurones de la couche précédente et de la couche suivante



56

## Exemple




Source https://www.etalab.gouv.fr/



57

## Paramètres du réseau

- Poids p de toutes les liaisons entre neurones et biais b de tous les neurones
- Déterminent le comportement du réseau
- Calculés lors de la phase d'apprentissage en utilisant les exemples d'apprentissage



58

## Calcul des paramètres

- 2 étapes pour chaque exemple (ou groupe d'exemples) d'apprentissage :
  1. Calcul des gradients de la fonction de perte par rapport aux paramètres, par rétropropagation (*back propagation*)
  2. Ajustement des paramètres par descente de gradient, pour minimiser la fonction de perte qui calcule la distance entre
     - la valeur calculée par le réseau
     - la valeur attendue pour l'exemple d'apprentissage



59

## Rétropropagation

- Fonction de perte L ; elle dépend des matrices $W_i$ des paramètres de la couche numéro i vers la couche numéro i + 1 (on suppose n couches cachées)
  - $\partial L/\partial W_{n+1}$ est d'abord calculé ($W_{n+1}$ contient les paramètres de la dernière couche cachée vers la couche de sortie)
  - puis $\partial L/\partial W_n$ est calculé en utilisant le résultat du calcul précédent
  - et ainsi de suite pour arriver à la fin au calcul de $\partial L/\partial W_1$ ($W_1$ représente les paramètres de la couche d'entrée vers la 1ère couche cachée)



60

## Théorème d'approximation universelle

- Rappel : un modèle d'IA peut être considéré comme une fonction $y = f(x_1, ..., x_n)$
- Théorème : un réseau de neurones peut approcher une fonction quelconque d'aussi près que l'on veut
- « Un réseau de neurones à une seule couche cachée, utilisant une fonction d'activation non linéaire appropriée (comme sigmoïde, tanh, ReLU…), peut approximer n'importe quelle fonction continue sur un compact de $R^n$, aussi précisément que l'on veut, à condition d'avoir suffisamment de neurones dans cette couche » ; https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d%27approximation_universelle



61

## Fonction approchée par un LM

- L'entrainement du LM va permettre de trouver les paramètres pour approcher une fonction qui
  - prend un texte en entrée
  - renvoie en sortie, pour chaque mot du vocabulaire, la probabilité que ce mot convienne pour compléter le texte (selon textes donnés lors de l'apprentissage du LM)
- La suite va préciser cette fonction



62

## Application à la génération de texte

- Le but d'un LM étant la génération de texte, la fonction approchée va être utilisée pour cette **fonction de base** :
  - prend un texte en entrée
  - renvoie en sortie un des mots les plus probables pour compléter le texte
- Pendant l'inférence, le LM va utiliser la fonction de base pour générer du texte **mot par mot**
- Par exemple pour répondre à une question ou pour générer la traduction d'un texte



63

## Traitement des textes

- On a vu que les réseaux de neurones font des calculs sur les données **numériques** en entrée pour calculer des données en sortie
- Comment les textes sont-ils traités par les réseaux de neurones ?
- Les textes sont représentés par des nombres en 2 étapes :
  1. Ils sont découpés en **tokens**
  2. Chaque token est ensuite transformé en un vecteur de nombres réels appelé **embedding**



64

## Token

- Unité minimale de texte traitée par un réseau de neurones
- Peut être un mot, une partie de mot, un signe de ponctuation
- Il existe aussi des tokens spéciaux ; par exemple les tokens de début et de fin de génération
- La décomposition d'un mot en tokens dépend du tokenizer, le logiciel qui effectue le découpage
- Exemple : le mot « smartphone » est décomposé en 2 tokens « smart » et « phone » par GPT-4o (voir https://platform.openai.com/tokenizer)



65

## Exemple réponse à une question

- Question « Quel est le plus grand océan du monde ? »

1. Le texte est découpé en tokens : « Quel », « est », …
2. Le modèle génère le 1er token de la réponse « **L** » pour compléter la question
3. Puis il génère le token « ' » qui complète « Quel est le plus grand océan du monde ? **L** »
4. Et ainsi de suite, pour compléter avec « L'océan Pacifique. »
5. Finalement, le token généré suivant est le token spécial qui indique que la génération est terminée



66

## Vocabulaire

- Ensemble des tokens utilisés dans une langue (ou dans plusieurs langues)



67

## Embeddings

- Chaque token est donc traduit en un vecteur de nombres réels, appelé embedding
- Le passage aux embeddings va permettre
  - d'effectuer des calculs mathématiques sur les données manipulées
  - mais aussi d'ajouter du sens au texte ; des tokens ayant un sens proche sont traduits par des embeddings proches (voir support suivant)



68

## ONNX

- Open Neural Network Exchange
- Format ouvert et open source pour représenter les modèles
- Permet d'échanger des modèles entre différents frameworks et environnements d'exécution
- Facilite l'interopérabilité entre les outils de DL comme PyTorch, TensorFlow, etc.



69

## Types principaux de réseaux de neurones

- De convolution (CNN ; Convolutional Neural Network)
- Récurrent (RNN ; Recurrent Neural Network)
- Transformeur



70

## CNN

- Type de réseau de neurones spécialement conçu pour traiter les **images**



71

## CNN - Filtres

- Analyse l'image par petites zones pour identifier des caractéristiques importantes (contours, texture, …) en effectuant une opération mathématique appelée convolution
- Un CNN passe l'image d'entrée à travers plusieurs filtres pour détecter des caractéristiques de plus en plus complexes : d'abord des contours et textures de base, ensuite des formes géométriques, des objets, des visages
- Ces caractéristiques sont enregistrées dans des cartes de caractéristiques



72

## CNN - Pooling

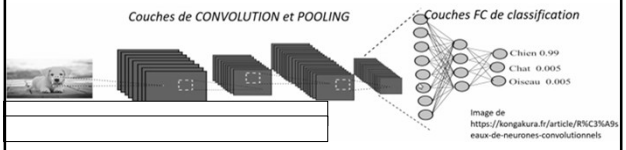
- Permet de réduire la taille des cartes de caractéristiques en conservant les informations les plus importantes
- Consiste à diviser l'image en plusieurs petites zones (par exemple 2 x 2 pixels) et à remplacer la zone par un seul point avec une seule valeur pour chacune des caractéristiques
- La valeur du point peut être le max, la moyenne, ou être obtenue par une autre opération à partir des valeurs des pixels de la zone



73

## CNN - Réseau de neurones

- Finalement, les informations extraites des filtres et du pooling sont envoyées en entrée d'un réseau de neurones pour donner une réponse ; par exemple pour savoir si une image représente un chien, un chat ou un oiseau






74

## Réseaux de neurones récurrents

- Les RNNs ont été introduits pour traiter les données ordonnées en séquences : texte, image, son,..
- Le traitement utilise une chaîne de réseaux de neurones (le même réseau est utilisé dans toute la chaîne) ; à chaque étape la sortie d'un réseau de la chaîne est donnée en entrée au réseau suivant
- Exemple : 1er mot d'une phrase est donné en entrée au réseau puis on donne en entrée au 2ème réseau la sortie du 1er réseau et le 2ème mot de la phrase, etc.



75

## Problèmes RNNs

- Traitement séquentiel des tokens, donc apprentissage long et moins performant pendant leur utilisation
- Difficulté à capturer les relations complexes entre les éléments de la séquence ou les relations entre éléments éloignés
- Biais temporel : les premiers éléments d'une séquence sont traités différemment des derniers
- Les transformeurs réduisent fortement ces problèmes



76

## Transformeurs



77

## Présentation

- Architecture pour traiter des séquences de données, composée de plusieurs modules dont certains utilisent des réseaux de neurones
- Peut prendre en charge plusieurs types de données : texte, image, vidéo, série temporelle,... ; dans la suite on se limitera aux textes
- Le module essentiel, appelée la **couche d'attention**, prend en compte les relations entre les éléments de la séquence, quelle que soit leur distance dans la séquence



78

## Motivation

- La génération du mot suivant par les LMs implique une bonne compréhension du texte à compléter
- Pour cela, il ne suffit pas de comprendre chaque mot isolément
- Il faut tenir compte de la syntaxe de la langue, du contexte de chaque mot et des relations entre les mots
- Les transformeurs peuvent extraire ces informations d'un texte, et les insérer dans les embeddings



79

## Exemple

- « Pierre a oublié ses lunettes sur le banc sur lequel il s'était assis. »
- Un transformeur, plus précisément la couche d'attention, va capturer la relation entre « banc » et « assis », bien qu'ils soient séparés (il a appris que ces mots sont souvent liés, parce qu'ils apparaissent souvent ensemble dans les phrases sur lesquelles on l'a entrainé)
- Il va aussi associer « Pierre » et « il »



80

## Structure d'un transformeur

- Essentiellement 2 parties : décodeur et encodeur
- En fait, une pile d'encodeurs et une pile de décodeurs
- Un transformeur peut n'utiliser qu'une de ces parties ; par exemple, GPT-4 n'utilise qu'une pile de décodeurs
- L'encodeur enrichit une séquence d'entrée en produisant une représentation contextuelle de chaque élément de la séquence Représentation contextuelle ?
- Le décodeur génère une sortie, élément par élément



81

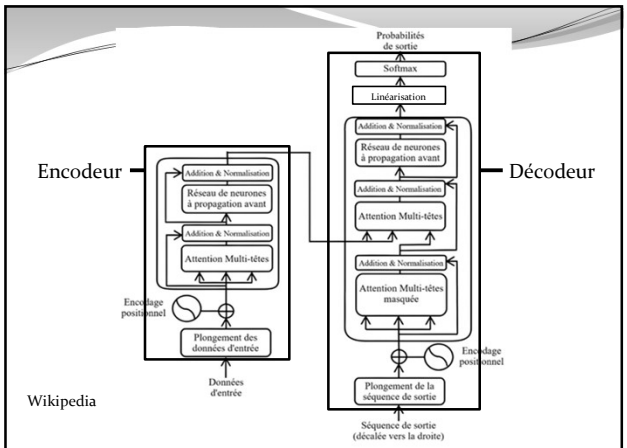
## Exemple : traduction de texte

- Traduire du texte « mot à mot » (plus exactement token par token) fournit une mauvaise traduction
- Il faut 2 étapes pour une traduction acceptable :
  1. Encodeur transforme $x=(x_1, ..., x_n)$ en $z = (z_1, ..., z_n)$ ; les $z_i$ sont enrichis par le contexte (relations avec les autres embeddings)
  2. Décodeur génère $y = (y_1, ..., y_m)$ à partir de z (le plus souvent $n \neq m$) ; processus autorégressif : $y_p$ est généré en utilisant les $y_i$ déjà générés ($i < p$), jusqu'à atteindre le token de fin

Exemple simple de traduction français-anglais avec $n \neq m$ ?



82

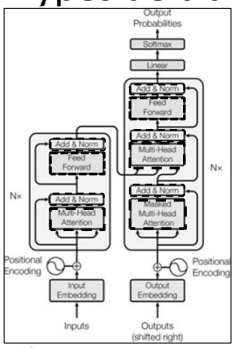



Wikipedia



83

## Types de traitement




- Couche d'attention
- Réseau de neurones (Feed Forward)
- Addition et normalisation
- Chacune de ces étapes modifie les embeddings



84

## Utilité des éléments (1/2)

- **Couche d'attention** : capture dans les embeddings les relations entre les tokens, indépendamment de leur position dans la séquence, et détermine les parties importantes de la séquence
- **Réseau de neurones** (Feed-forward) : l'étape de l'attention est linéaire (assez rigide) ; le réseau de neurones utilise sa non-linéarité pour apprendre des informations plus complexes et enrichir chaque embedding indépendamment



85

## Utilité des éléments (2/2)

- L'addition et la normalisation se font après chaque traitement
  - **Addition** : les données en entrée sont ajoutées aux données en sortie, ce qui évite qu'elles soient noyées dans les nouvelles données
  - **Normalisation** : évite des problèmes techniques durant l'apprentissage en réduisant les différences de grandeur entre les valeurs calculées ; évite en particulier « l'explosion » et « l'extinction » des gradients



86

- La suite détaille les processus qui s'exécutent dans les transformeurs est destinée à ceux qui veulent en savoir plus, mais elle ne fait pas partie du programme à connaitre pour l'examen



87

## Encodeur

- Transforme la séquence d'entrée (par exemple un texte qui représente une question, ou un texte à traduire) en une séquence d'embeddings
- Les embeddings sont traités en parallèle
- Utilise un mécanisme d'**attention** qui permet de s'intéresser plus particulièrement à certaines parties de la séquence d'entrée et de détecter les relations entre ces parties, même si elles sont éloignées dans la séquence
- Le résultat est une représentation enrichie de la séquence d'entrée (même nombre d'embeddings)



88

## Décodeur

- De base, prend une séquence de tokens et sort le token suivant (itérativement une séquence de tokens)
- Si traitement préalable par encodeur, utilise la représentation enrichie de la séquence d'entrée fournie par l'encodeur pour générer une séquence de sortie
- Génère la séquence de sortie token par token, en se basant sur les informations fournies par l'encodeur, et sur les tokens qu'il a déjà générés
- Pour chaque itération, les probabilités de tous les tokens du vocabulaire sont calculées et le transformeur en choisit un (le choix dépend des hyperparamètres)



89

## Encodeur ou/et décodeur

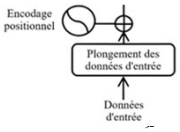
- Peuvent être utilisés indépendamment
- Modèles uniquement encodeurs : pour tâches qui nécessitent une compréhension de l'entrée, comme classification de phrases, analyse de sentiments, recherche de mots masqués
- Modèles uniquement décodeurs : pour tâches génératives telles que la génération de texte
- Modèles encodeurs-décodeurs : pour tâches génératives qui nécessitent une entrée, telles que la traduction ou le résumé de texte



90

## Travail préparatoire pour encodeur et décodeur

1. Séquence d'entrée est découpée en tokens
2. Chaque token est transformé en embedding, vecteur numérique de grande dimension
3. Tous les embeddings sont traités en parallèle ; l'encodage positionnel ajoute aux embeddings des informations sur la position des tokens






91

## Encodage positionnel

- Un grand avantage pour l'efficacité des transformeurs par rapport aux réseaux récurrents est qu'ils peuvent traiter en parallèle tous les embeddings en entrée
- Pour ne pas perdre la position de chaque token dans la séquence, cette information est codée dans un vecteur de même dimension que l'embedding, et ajoutée à l'embedding



92

## Addition et normalisation

- Chaque étape importante des encodeurs et décodeurs est suivie d'une étape d'addition et de normalisation
- **Addition** : ajoute aux embeddings des informations fournies par l'étape précédente (par exemple la couche d'attention) pour que cette information ne soit pas « oubliée » car déformée par la dernière étape
- **Normalisation** : il peut y avoir des variations importantes d'amplitude dans les valeurs fournies par les différentes étapes, ce qui peut fausser le travail effectué dans les étapes suivantes ; la normalisation contrôle ces variations



93

## Partie 1 : encodeur

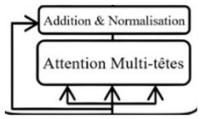
Enrichit la séquence d'entrée



94

## Multi-head (ou self-) attention

- Etape très importante qui permet à l'encodeur de se concentrer sur les embeddings importants de la séquence d'entrée et de capturer à quel point chaque embedding est relié à chaque autre embedding
- A la sortie les embeddings sont enrichis avec l'attention






95

## Utilité de l'étape d'attention

- Sens des mots :
  - « Je n'ai pas pu aller à mon cours de batterie car la batterie de ma voiture est tombée en panne »
  - La seule façon de distinguer les 2 sens du mot « batterie » est d'analyser le contexte qui l'entoure ; l'attention le fait (relation de « batterie » avec « cours » ou bien avec « voiture »)
- Relation entre les mots :
  - « Jean a montré à Paul où il devait signer. »
  - L'attention va permettre d'associer « il » à Paul



96

## Pourquoi « plusieurs têtes » ?

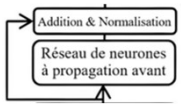
- On peut prêter attention à plusieurs choses dans une phrase
- De plus, une phrase peut être ambiguë :
  « Robert a vu Bernard avec le télescope »
  peut avoir 2 sens
- Une tête d'attention va associer « Robert », « a vu » et « Bernard », alors qu'une autre tête va associer « Bernard » et « avec le télescope »
- Les différentes têtes peuvent identifier différentes relations entre les mots ; les embeddings capturent ces différentes interprétations qui pourront être résolues par la suite avec le contexte



97

## Réseau de neurones à propagation avant

7. Feed-forward layer : les embeddings contextuels (enrichis) sont transmis à un réseau de neurones
8. Le réseau de neurones permet d'extraire les caractéristiques de plus haut niveau, grâce à ses capacités linguistiques et à ses connaissances générales






98

## Partie 2 : décodeur

Génère la séquence de sortie, token par token, à partir de la séquence d'entrée enrichie par le travail de l'encodeur



99

## Globalement

- Au début du traitement, la séquence de sortie du décodeur est vide (en fait, elle contient un token spécial de début de séquence) ; elle se remplit au fur et à mesure du traitement
- Celui-ci génère alors le 1er token qui est ajouté à la séquence de sortie du décodeur et on revient au début du traitement du décodeur
- Le décodeur génère ensuite le 2ème token, ajouté lui aussi à la séquence de sortie, et ainsi de suite, jusqu'à ce que le décodeur génère un token d'arrêt



100

## Masked multi-head attention

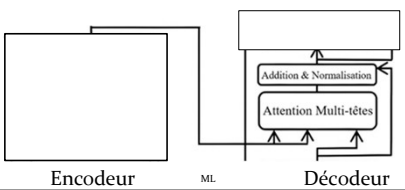
- « Self-attention »
- Comme pour l'encodeur, ajoute des informations d'attention sur la séquence de sortie déjà générée pour comprendre les relations entre ses tokens
- « Masked » car, pendant l'entrainement, on masque les tokens qui suivent le dernier token généré pour simuler ce qui va se passer pendant l'inférence : à ce stade de la génération, on ne connait pas les tokens à droite du dernier mot généré
- Par souci de consistance, cette étape est conservée pendant l'inférence




101

## Multi-head attention

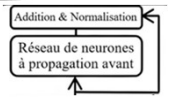
- « Cross-attention » ; différente de « self-attention » car utilise la séquence de sortie déjà générée **et** la séquence d'entrée enrichie par l'encodeur (étape éliminée si pas d'encodeur)
- Enrichit les embeddings de la sortie du décodeur en intégrant ceux de l'entrée ; particulièrement utile pour les tâches de traduction






102

## Réseau de neurones




- En entrée, les embeddings enrichis par la couche d'attention
- La couche de sortie a autant de neurones que la dimension des embeddings utilisés
- Les valeurs de tous ces neurones de la couche de sortie décrivent à quoi va ressembler le prochain token (représentation riche et contextuelle de ce token)
- Les étapes suivantes vont permettre de calculer la distribution de probabilité des éléments du vocabulaire



103

## Linéarisation

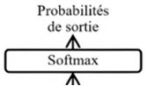

- Prend en entrée l'embedding h, issu de la sortie du dernier bloc feed forward ; h est une représentation riche de l'embedding du token qui est attendu comme prochain token
- A partir de h, une matrice (apprise par rétropropagation), dite matrice de projection, permet de calculer le score (*logit*) de chaque élément du vocabulaire pour être le prochain token



104

## Softmax, probabilités de sortie

- Softmax : Fonction qui génère des probabilités à partir des logits; indique les chances que le modèle choisisse chaque token comme prochain dans la séquence
- Prédiction prochain token peut être faite de plusieurs manières :
  - Choisir le token le plus probable
  - Choisir le token parmi les plus probables (le choix aléatoire se fait en tenant compte des probabilités)






105

## Références



106

## IA pour les décideurs

- Présentation de l'IA pour les décideurs (2021 ; UNESCO): https://unesdoc.unesco.org/ark:/48223/pf0000380006



107

## MOOCs gratuits

- L'Intelligence Artificielle… avec intelligence ! : https://www.fun-mooc.fr/fr/cours/lintelligence-artificielle-avec-intelligence/
- Intelligence artificielle pour et par les enseignants : https://www.fun-mooc.fr/fr/cours/intelligence-artificielle-pour-et-par-les-enseignants-ai4t/



108

- Lexique IA : https://www.editions-eni.fr/lexique-intelligence-artificielle-chatgpt
- Teachable machine de Google pour créer des modèles de ML rapidement et simplement, avec des images ou de l'audio, ou des poses (par exemple lever le bras) : https://teachablemachine.withgoogle.com/



109

- Lexique IA : https://www.editions-eni.fr/lexique-intelligence-artificielle-chatgpt
- Teachable machine de Google pour créer des modèles de ML rapidement et simplement, avec des images ou de l'audio, ou des poses (par exemple lever le bras) : https://teachablemachine.withgoogle.com/



110

## Vidéo sur IA

- Vidéo de 2 h 45 très intéressante de Yann Lecun sur l'IA en général et les limites des LLMs, en particulier des modèles non open source : https://www.youtube.com/watch?v=5t1vTLU7s40



111

## Vidéos pour machine learning

- https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw ; StatQuest essaie de vulgariser les statistiques et le machine learning
- https://www.youtube.com/watch?v=trWrEWfhTVg de David Louapre, sur le deep learning
- Tour d'horizon des modèles et algorithmes en 2 vidéos, avec des descriptions rapides de chaque modèle et algorithme ; Machine Learnia, Guillaume Saint-Cirgue : https://www.youtube.com/watch?v=mT6NnslbNLM



112

## Vidéos pour réseaux de neurones

- https://www.youtube.com/watch?v=7ell8KEbhJo, en français ; à voir pour commencer ; David Louapre
- https://www.youtube.com/watch?v=XUFLq6dKQok (Machine learnia), en français, une série pour étudier les algorithmes et mathématiques utilisés par IA



113

- Cours coursera sur deep learning : https://www.coursera.org/specializations/deep-learning (accès gratuit pendant 7 jours seulement)
- Hugging Face, site Web pour ceux qui travaille avec le deep learning : https://huggingface.co/



114

- https://www.youtube.com/watch?v=rn1EEd6M7fA : courte vidéo qui explique rapidement le deep learning et qui utilise TensorFlow (librairie qui permet de réaliser le réseau de neurones) et Keras (librairie pour faire du deep learning en entrainant l'IA) ; fait partie de la série « Informatique sans complexe » qui contient d'autres vidéos courtes sur l'IA et sur divers autres domaines de l'informatique : https://www.youtube.com/@InformatiqueSansComplexe



115

- Site Web Kaggle pour une communauté d'utilisateur de IA et de ML (Machine Learning). Pour apprendre (cours, guides), pour les développeurs, pour les chercheurs, et des projets sous la forme de compétitions ou autres formes. https://www.kaggle.com/
- Article de vulgarisation très intéressant sur le phénomène de « grokking » : https://scienceetonnante.substack.com/p/grokking-les-modeles-dia-sont-ils



116

## Transformeurs (1/2)

- Cours Hugging Face : https://huggingface.co/learn/nlp-course/chapter1/1
- Visual Guide to Transformer Neural Networks : https://www.youtube.com/playlist?list=PL86uXYUJ7999zE8u2-97i4KG_2Zpufkfb
- Le transformer illustré : https://a-coles.github.io/2020/11/15/transformer-illustre.html (en français), https://jalammar.github.io/illustrated-transformer/, en vidéo : https://www.youtube.com/watch?v=-QH8fRhqFHM
- https://lbourdois.github.io/blog/nlp/Transformer/



117

## Transformeurs (2/2)

- Article fondateur des transformeurs « Attention is all you need », https://arxiv.org/abs/1706.03762
- Article « The annotated transformer » qui illustre l'article avec du code en Python : https://nlp.seas.harvard.edu/2018/04/03/attention.html
- Vidéo sur transformers, par Batool Haider : https://www.youtube.com/playlist?list=PL86uXYUJ7999zE8u2-97i4KG_2Zpufkfb
- Article de Anthropic (LLM « Claude ») sur la « pensée » des LLMs : https://www.anthropic.com/news/tracing-thoughts-language-model



118