



Le réseau de neurones qui écrivait des romans

Lab BDX/IO 2022

BDX I/O

onepoint.

Au-delà de l'évidence

Vos accompagnatrices

Bérengère MATHIEU

- Master en informatique, spécialité Analyse d'Image
- Doctorat dans le domaine de l'image et de l'intelligence artificielle
- 5 ans d'expérience dans le milieu professionnel
- Enseignante/Formatrice depuis 10 ans

Cécile HANNOTTE

- Diplôme d'ingénieur en Mathématiques et Informatique à l'INSA de Rouen
- En parallèle M2 en Science des données obtenu
- 2 ans d'expérience dans le milieu professionnel

George Sand

- Amantine Aurore Lucile Dupin de Francueil
- Écrivaine, journaliste, militante
- Passionnée par les sciences naturelles et avant-gardiste sur les questions écologiques



Partant pour l'aventure ?

Votez pour l'image qui représente le mieux votre état d'esprit



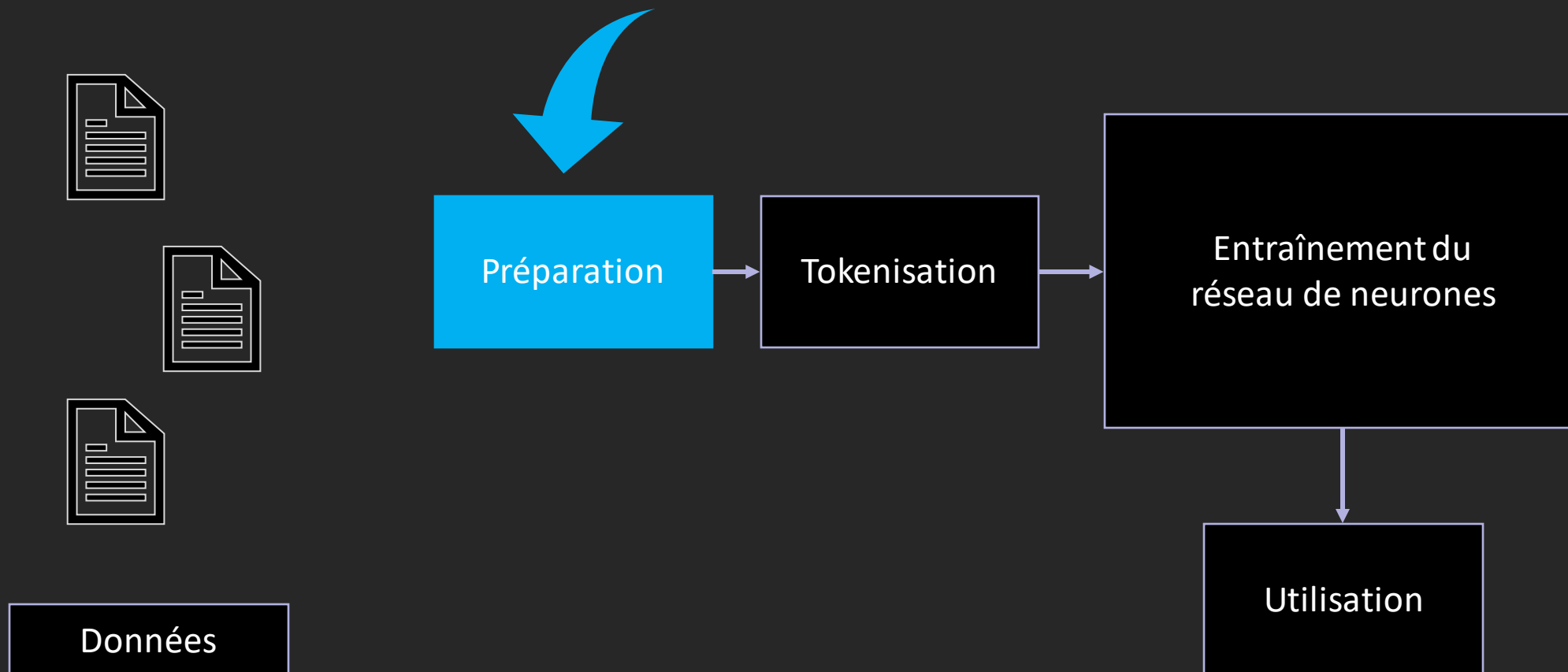
Au programme

1. Présentation
2. Préparer vos données
3. Tokenisation
4. Spécialisation
5. Utilisation
6. Conclusion

Préparer vos données

*La mare au diable de la data science,
ce sont les données !*

Vous êtes ici

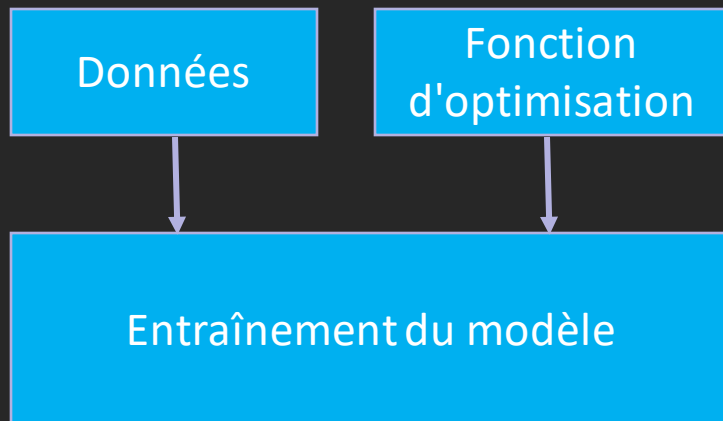


Les fondations pour un projet solide

- Est-ce que je dispose d'assez de données ?
 - Pour une même problématique différentes solutions sont possibles
 - Pour chaque solution le nombre minimal de données nécessaire doit être évalué
 - Ne pas avoir ce nombre de données, c'est faire courir un risque au projet
- Est-ce que je dispose de la vérité terrain ?
 - Vérité terrain : la solution attendue
 - Les réseaux de neurones sont des techniques par apprentissage supervisé (supervised machine learning)
 - Sans la vérité terrain le réseau ne peut pas apprendre et le projet est bloqué

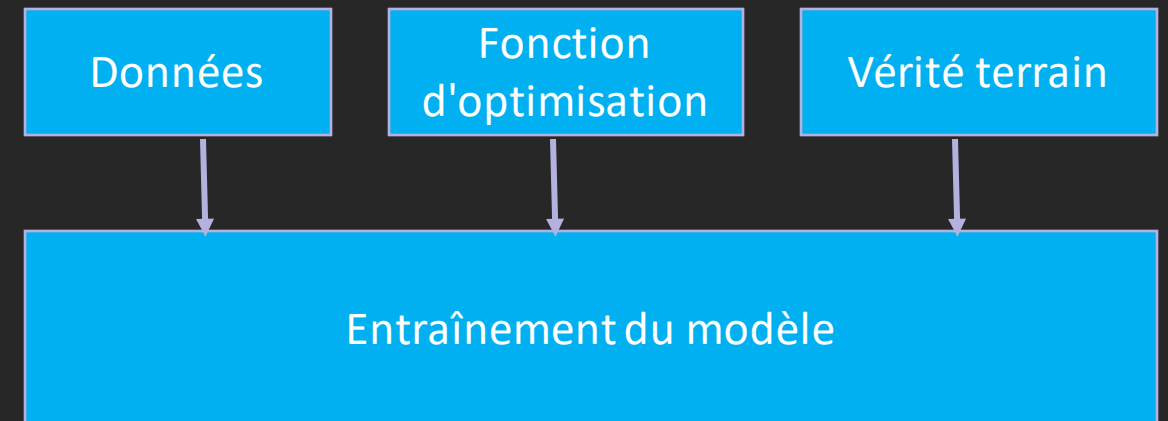
Deux techniques d'apprentissage

NON-SUPERVISE



Exemples : identifier des groupes cohérents de profils utilisateurs, compresser une image, etc.

SUPERVISE



RN = Apprentissage supervisé uniquement!

Exemples : générer des images réalistes, localiser et identifier un élément, traduire un texte, etc.

Les fondations pour un projet solide



Mettre en place une démarche scientifique

- Le réseau est **entraîné** à réaliser une tâche **sur une partie des données**. Pour savoir si l'apprentissage est concluant nous devons **l'évaluer** sur des **données** qu'il n'a **jamais utilisées**
- Fidélité des résultats : si j'utilise des **données** d'apprentissage et d'évaluation **différentes**, est-ce que j'obtiens des **scores de réussite similaires** ?
- La **validité** de notre expérience est **limitée** : savoir générer du George Sand n'implique pas que notre réseau saura générer du Virginia Woolf.

Les fondations pour un projet solide

Mettre en place des outils de qualité de code

- Utiliser un IDE : éviter les Notebooks
- Documenter : fonctions, processus de traitement des données, utilisation du code, etc.
- Respect des standards de Python : [Pylint](#)
- Ecrire des tests : unitaires, de non-régression pour les codes de haut niveau

Mise en place de l'atelier (voir le README.md)

1. Récupérer le code sur Github : <https://github.com/channotte/text-gen>
2. Cloner le projet et ouvrir le dossier récupéré
3. Se créer un compte Hugging Face (<https://huggingface.co/>) et générer un token d'accès
4. Renseigner le token et son pseudo dans le fichier aurore/credentials.ini
5. Ouvrir docker et construire le container docker en lançant : **docker build -t text-gen** .
6. Lancer le docker pour vérifier que tout se passe bien
7. Partager vos ressources sur <https://mensuel.framapad.org/p/5zjchxuzss-9xx2?lang=fr>

Linux : `docker run -v $PWD/aurore:/aurore text-gen python aurore/1_prepare_dataset_solution.py`

Windows : `docker run -v <chemin complet dossier text-gen>/aurore:/aurore text-gen python aurore/1_prepare_dataset_solution.py`

Nous allons utiliser un **volume** pour pouvoir modifier directement le code et les données contenues dans le docker, sans avoir à le reconstruire à chaque fois.

Les fondations pour un projet solide

Exercice 1

- Nous allons utiliser un modèle de type GPT2 (*Generative, Pretrained transformer*)
- Regardez rapidement combien de données ont été nécessaires pour entraîner ce type de réseau de neurones

Exercice 2

- A votre avis, dans notre cas, en quoi consiste la vérité terrain ?

Exercice 3

- Quelle stratégie utiliseriez-vous pour sélectionner les données d'évaluation ?

Les fondations pour un projet solide

Est-ce que je dispose d'assez de données ?

- Méthode retenue : modèle de type GPT 2
- Données nécessaires : contenu d'environ 8 livres pour l'apprentissage et 1 pour le test
- Les livres de George Sand appartiennent au domaine public : projet Gutenberg.

Est-ce que je dispose de la vérité terrain ?

- Cas particulier où les données contiennent déjà la vérité terrain

Evaluation

- Nous pourrions sélectionner 1/10 des phrases de chaque livre
- Ou sélectionner un livre particulier

Récupérer les données

Le côté obscur de la force : utiliser une méthode de chargement de données trop générique

```
from dataset import load_dataset
lelia_url = "https://www.gutenberg.org/files/39738/39738-0.txt"
la_petite_fadette_url = "https://www.gutenberg.org/cache/epub/34204/pg34204.txt"
gabriel_url = "https://www.gutenberg.org/cache/epub/13380/pg13380.txt"
lettre_voyageur_url = "https://www.gutenberg.org/files/37989/37989-0.txt"
la_mare_au_diable_url = "https://www.gutenberg.org/files/23582/23582-0.txt"

train_paths = [lelia_url, la_petite_fadette_url, gabriel_url, lettre_voyageur_url]
test_path = la_mare_au_diable_url

dataset = load_dataset("text", data_files={"train": train_paths, "test": test_path})
```

Inclus les 50 premières lignes qui n'ont rien à voir avec le texte de George Sand mais sont des ajouts du projet Gutenberg

Exercice 1 : récupérer les données

Nous allons compléter le code pour préparer les données

- Méthode **download_file** : pour chaque fichier
 1. Utilisez la fonction [get_file](#) de la bibliothèque Keras pour récupérer chacun des fichiers
 2. Ouvrir le fichier avec la fonction Python [open](#)
 3. Récupérer la liste des lignes contenues dans ce fichier avec la méthode `readlines()`
 4. Retirer les 50 premières lignes
 5. Retirer les 100 dernières lignes

Exercice 2 : Séparer les phrases

Nous allons compléter le code pour préparer les données

- Méthode `split_text_to_list` : identifier les phrases présentes dans chaque ligne
 1. Retirer les espaces en fin de ligne avec la fonction Python `rstrip`
 2. Recréer le texte original avec la méthode `join`
 3. Identifier le début et la fin de chaque phrase en utilisant la bibliothèque spécialisée `NLTK`

Exercice 3 : **calcul du** **nombre de** **mots**

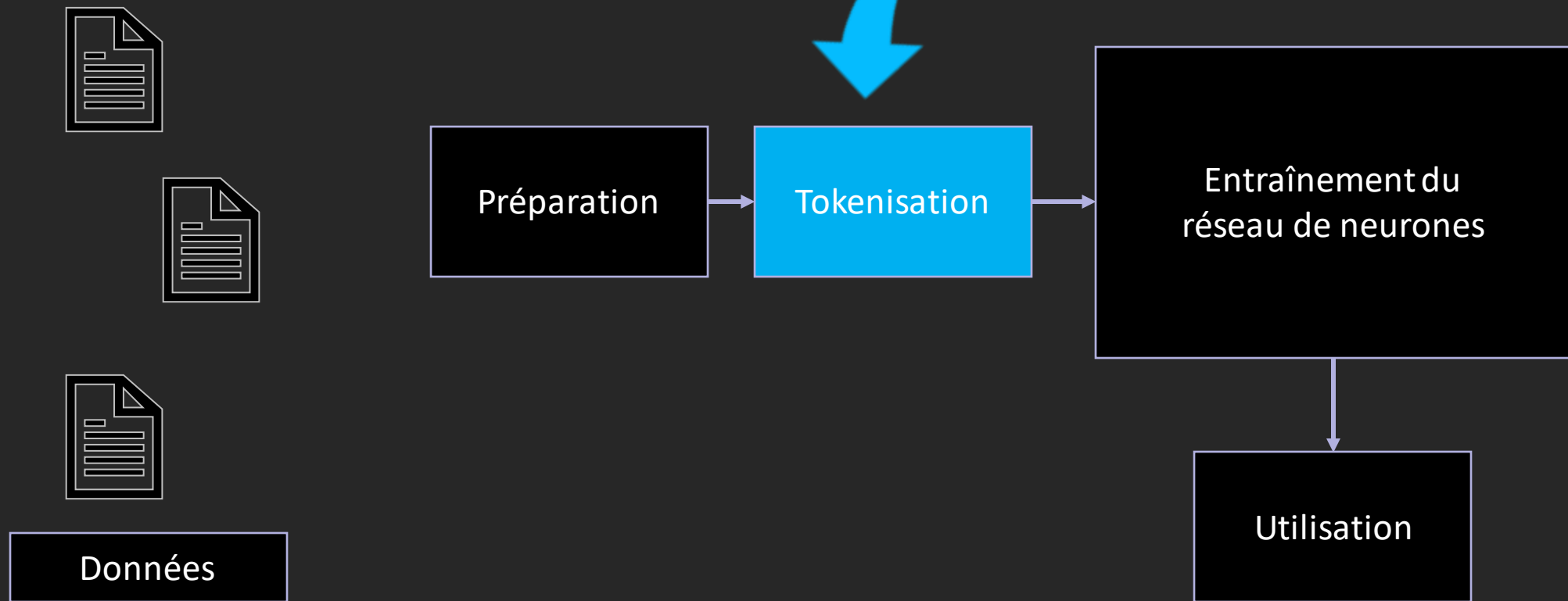
Quelques ajouts bien utiles :

- Afficher le nombre de phrases pour les données d'apprentissage et d'évaluation
- Regarder au hasard quelques phrases et vérifier qu'il n'y a pas trop d'erreurs

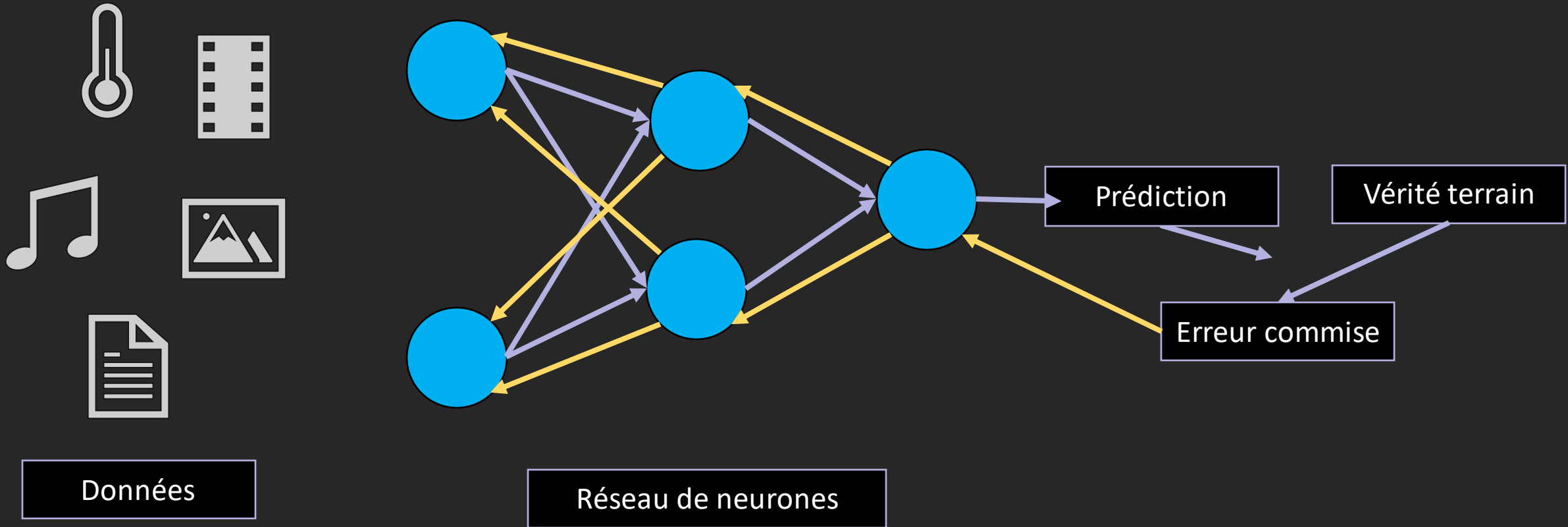
Tokenisation

Toutes les données sont complexes.
Certaines le sont plus que d'autres

Vous êtes ici

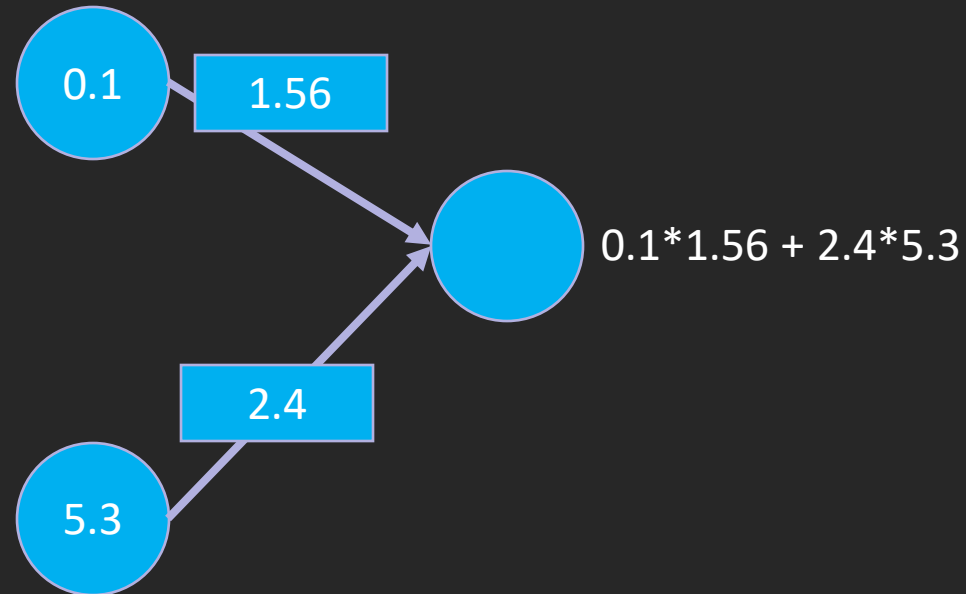


Entraînement d'un réseau de neurones



Entraîner un réseau de neurones

- Chaque neurone calcule la **somme pondérée** des entrées qu'il reçoit
- **Entraîner** un réseau de neurones revient à trouver les **bonnes valeurs** pour les **pondérations**
- Nos données doivent donc être sous **forme de nombres**

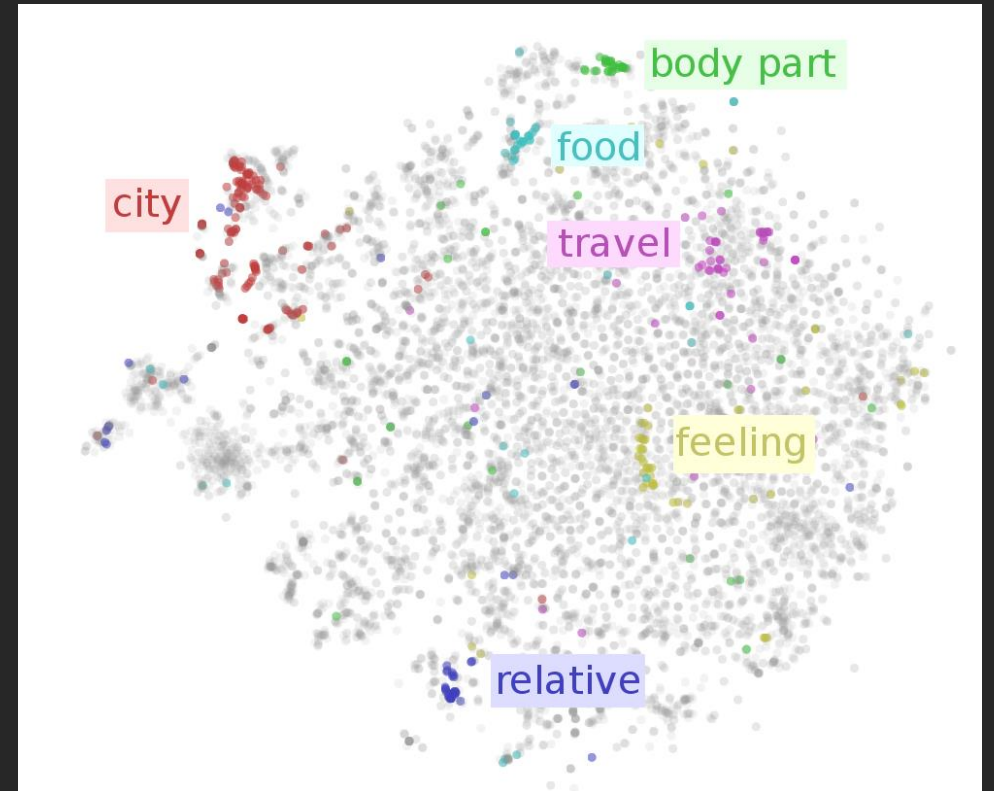


Convertir du texte en nombre

De nombreuses méthodes ont été proposées

Sac de mots : décrire un texte par le nombre d'occurrences de chaque mot du lexique

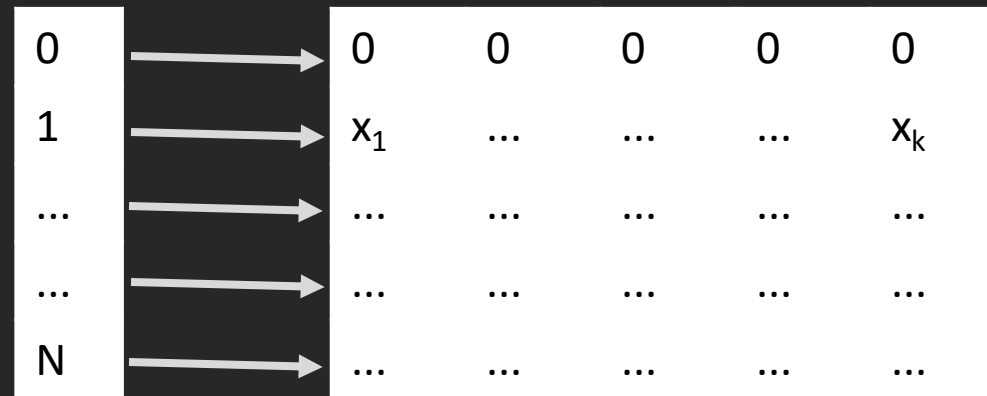
Embedding : remplacer chaque mot du lexique => un vecteur numérique



Source : [On word embedding](#), S. Ruder

Couche d'embedding dans les réseaux de neurones

Transforme des entiers positifs en vecteurs réels



Convertir du texte en nombre

- Pour les réseaux de neurones la première étape consiste à découper le texte en tokens et à associer un identifiant numérique à chaque token
- Un token peut être :
 - Une lettre
 - Une syllabe
 - Un groupe de lettre (un peu moins ou un peu plus qu'une syllabe)
 - Un mot
 - Un groupe de mot

Bien choisir ses tokens

- Tokens = comment le RN voit le texte
- Un token par lettre revient à épeler chaque lettre d'un texte et vous demandez d'un saisir le contenu ou de savoir en produire la suite
- Si le token font intervenir des groupes de lettres ou des mots, ils doivent être choisi en fonction du champ lexical du domaine cible (exemple avec CUTIE pour la comptabilité)

Bien choisir ses tokens

- Les **tokens** forment un **lexique fixe** : les RN ne savent pas gérer les tokens inconnus
- Pour être rigoureux il faut donc **créer les tokens à partir des données d'apprentissage uniquement**
- Nous pourrions ensuite vérifier leur pertinence pour les données d'évaluation

Exercice 1 : instancier un tokenizer

Télécharger le tokenizer pré-entraîné
benjamin/gpt2-wechsel-french grâce à la méthode
`from_pretrained` de la classe `AutoTokenizer`

Exercice 2 : tester le tokenizer

- Créer une variable contenant un texte simple, par exemple "Bonjour Madame, je m'appelle George Sand. Et vous ?"
- Analyser ce texte avec le tokenizer

```
tokens_analysis = pretrained_tokenizer(txt)
```

Tokens_analysis est un dictionnaire et la valeur associée à la clé **input_ids** donne le découpage en tokens

- Utiliser la fonction [convert_ids_to_tokens](#) pour afficher le texte associé à chaque token

Exercice 3 : spécialiser le tokenizer

- Utiliser la fonction `get_training_corpus` pour récupérer un itérateur sur le jeu de données d'entraînement
- Utiliser la méthode `train_new_from_iterator` du tokenizer pour spécialiser le tokenizer avec comme paramètre :
 - `text_iterator` : le résultat de la fonction `get_training_corpus`
 - `vocab_size` : 52 000

Exercice 3 : sauvegarder le tokenizer

- Utiliser la fonction [save_pretrained](#) pour sauvegarder le tokenizer dans le répertoire aurore/tokenizer
- Observer le résultat

Documentation complémentaire

[Hugging Face: Understanding tokenizers](#)



Tokenizers

Spécialisation

*Sélectionner le bon candidat
pour résoudre notre problématique*

Vous êtes ici



Données



Préparation

Tokenisation

Entraînement du
réseau de neurones

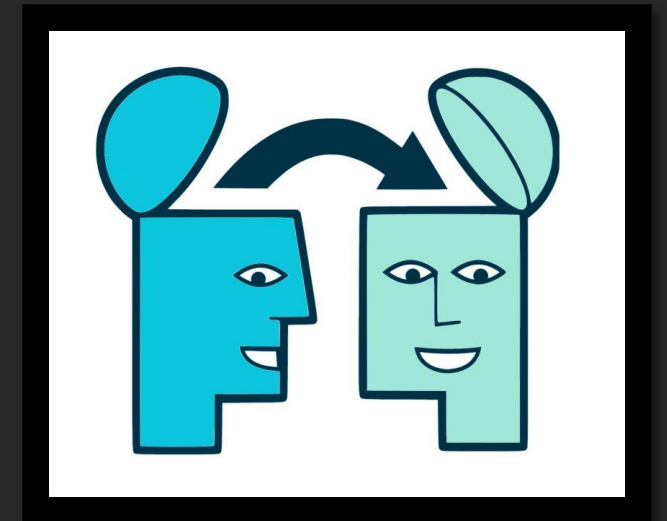
Utilisation

Comment choisir un réseau de neurones

- Veille scientifique et technique sur la thématique cible (ici la génération de texte)
- S'assurer que vos données sont proches des données utilisées. (ici validation sur données littéraires, si possible en français)
- Est-ce que du code est disponible ? Si oui est-il de qualité ? 
- Est-ce qu'un modèle pré-entraîné est disponible ? 

La Spécialisation

- Spécialiser un modèle consiste à partir d'un réseau de neurones déjà entraîné et à **poursuivre cet apprentissage** sur de nouvelles données
- Repose sur le principe qu'il est plus facile d'apprendre à jouer du violon si vous maîtrisez déjà le piano et la guitare que si c'est votre premier instrument
- En anglais on parle également de **transfert learning**
- Les dernières innovations pour les réseaux de neurones sont si coûteuses à entraîner de zéro (énergie et données) que la majorité des entreprises se contentent de les spécialiser



Les catalogues de modèles



Hugging Face

<https://huggingface.co/>

Model Zoo

<https://modelzoo.co/>

Mettent à disposition des réseaux de neurones déjà entraînés, classés par thématiques et par frameworks.

Quels modèles pour la génération de texte

Modèle prenant en compte la temporalité

Réaliser la prédiction présente en prenant en considération les prédictions passées.

Je génère un nouveau mot en me basant sur les mots que j'ai déjà proposé.

Auto-encoder

Apprendre à encoder le texte via un vecteur numérique et à décoder ce vecteur pour retrouver le texte.

Décoder des vecteurs générés avec une dose d'aléatoire.

Modèle avec mécanisme d'attention

Réaliser les prédictions en observant globalement le texte et en ciblant ce qui est pertinent.

Le mot que je génère est davantage lié au sujet et au verbe de ma phrase qu'aux déterminants

GPT-2 : le choix pour ce codelab

- Conçu par OpenAI
- GPT : Generative Pre-trained Transformer
- Principe : entraîner un réseau sur un corpus très large (des centaines de milliards de données) avec une tâche où la vérité terrain peut être générée automatiquement (apprentissage non-supervisé)
- Ce réseau peut ensuite être spécialisé sur des tâches plus complexes, nécessitant de la vérité terrain
 - L'apprentissage est plus rapide
 - La vérité terrain nécessaire est moins importante

GPT-2 : bibliographie

- Présentation des modèles de type transformer : [Attention is all you need](#)
- Présentation des modèles de type GPT : [Improving language understanding by generative pre-training](#)
- Polémique à la sortie de GPT-2 : [GPT-2 d'OpenAI : Un meilleur outil de traitement automatique du langage et les questions éthiques qu'il soulève](#)
- Article de recherche associé : [Language Models are unsupervised multitask learner](#)
- [Présentation sur Huggingface](#)

Bloom : l'avenir pour la génération de texte ?

- Encore un modèle de type **transformer**
- Réalisé par une équipe **internationale** (50 pays différents, une vingtaine de langages couverts)
- Conçu pour respecter une charte **éthique** (éviter les discriminations, transparence, entraînement éco-responsable etc.)
- [A New BLOOM in AI? Why the BLOOM Model Can Be a Gamechanger](#)
- [Understand BLOOM, the Largest Open-Access AI, and Run It on Your Local Computer](#)

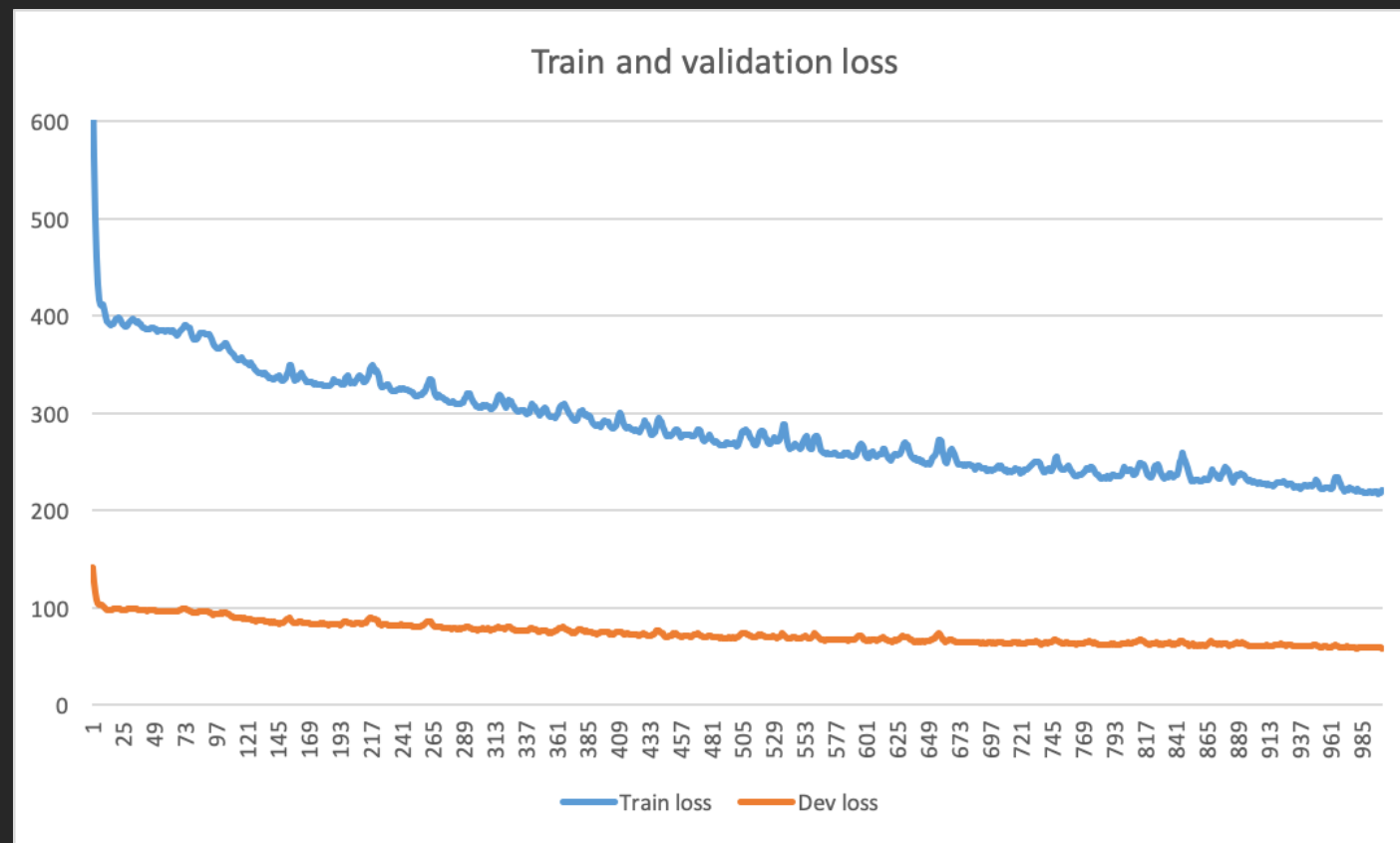
Exercice 1 : Tokenisation

1. Charger le tokenizer spécialisé avec la méthode [from_pretrained](#) de la classe `AutoTokenizer`
2. Charger le jeu de données avec la fonction [load_from_disk](#)
3. La fonction `map` nous permet de convertir efficacement nos données en tokens, grâce au calcul parallèle

```
tokenized_datasets = dataset.map(  
    tokenize, batched=True,  
    remove_columns=dataset["train"].column_names  
)
```

Apprentissage par lot

- Un réseau de neurone apprend en **itérant plusieurs fois sur les données** d'apprentissage et sur la vérité terrain
- Lorsque le réseau de neurones a parcouru l'ensemble des données d'apprentissage on dit qu'il a accompli une **époque**

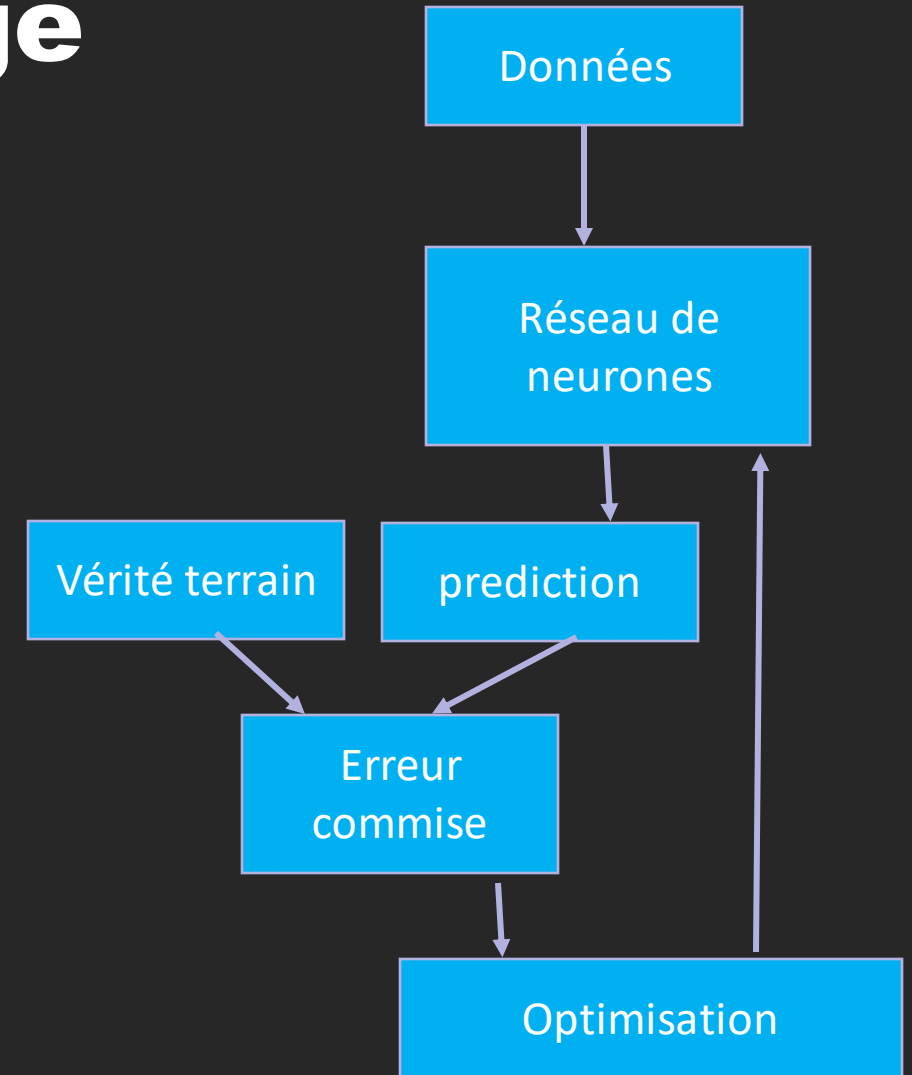


Source : [Why my network needs so many epochs to learn?](#)

Stratégie d'apprentissage

Pour apprendre vite et bien je dois déterminer quand est-ce que je calcule l'erreur commise et que je met à jour mon réseau

- **Stratégie 1** : à chaque donnée => entraîne beaucoup d'instabilité, risque de partir dans un minima local
- **Stratégie 2** : à chaque époque => lent
- **Stratégie 3** : à chaque lot (batch) de N données



Exercice 2 : chargement par lot

- Créer une instance de la classe [DataCollatorForLanguageModeling](#) avec comme paramètres : le tokenizer, mlm=False, return_tensors="tf"
 - Cet objet va nous permettre de charger nos données par lot
- La variable **tokenized_datasets** permet d'accéder aux deux jeux de données (apprentissage et validation) mais comme notre réseau de neurones est créé avec Tensorflow il faut convertir chaque jeu de données en un "dataset tensorflow" grâce à la méthode [to_tf_dataset](#)
- C'est cette fonction qui va faire le lien entre le jeu de données et le DataCollator (l'assembleur de données)

Exercice 3 : créer un modèle pré- entraîné

1. On commence par créer une configuration pour notre réseau de neurones
2. Ce sont ces paramètres qui vont nous permettre de récupérer une version spécifique du réseau de neurones GPT-2

```
config = AutoConfig.from_pretrained(  
    "gpt2",  
    vocab_size=len(tokenizer),  
    n_ctx=context_length,  
    bos_token_id=tokenizer.bos_token_id,  
    eos_token_id=tokenizer.eos_token_id,  
)  
model = TFGPT2LMHeadModel(config)
```

Exercice 4 : entraînement

1. Nous choisissons une méthode d'optimisation adaptée : elle adapte les pondérations du réseau pour diminuer petit à petit l'erreur commise. Nous utilisons la fonction [create_optimizer](#). Vous pouvez voir qu'il s'agit d'une méthode d'optimisation de type [ADAM](#)
2. Ensuite nous associons le réseau de neurones et la méthode d'optimisation via la méthode [compile](#) du réseau de neurones
3. Enfin nous pouvons lancer l'apprentissage via la méthode [fit](#) du réseau de neurones

Pour des raisons de temps de calcul, nous ne pourrons entraîner que durant une seule époque.

Utilisation

*Intégrer notre réseau de neurones
au sein d'une application métier*



Données

Préparation

Tokenisation

Entraînement du
réseau de neurones

Utilisation

Vous êtes ici :



Utiliser un réseau de neurones

- Un RN n'est jamais que l'une des briques d'un projet beaucoup plus complexe
- Il vous faudra **préparer et vérifier les données** fournies par l'utilisateur (le texte est-il en français ? Contient-il des propos injurieux ? Etc.)
- Il vous faudra également **mettre en forme le résultat** produit par le réseau de neurones
- Vous devrez gérer des problématiques d'accès, de mises à jour, etc.

L'importance de l'expérience utilisateur

- Même le meilleur réseau de neurones ne peut se passer d'une interface utilisateur pensée pour être intuitive et confortable
- Cette interface permet d'adapter le réseau de neurones au fonctionnement de l'utilisateur et non l'inverse

Flask



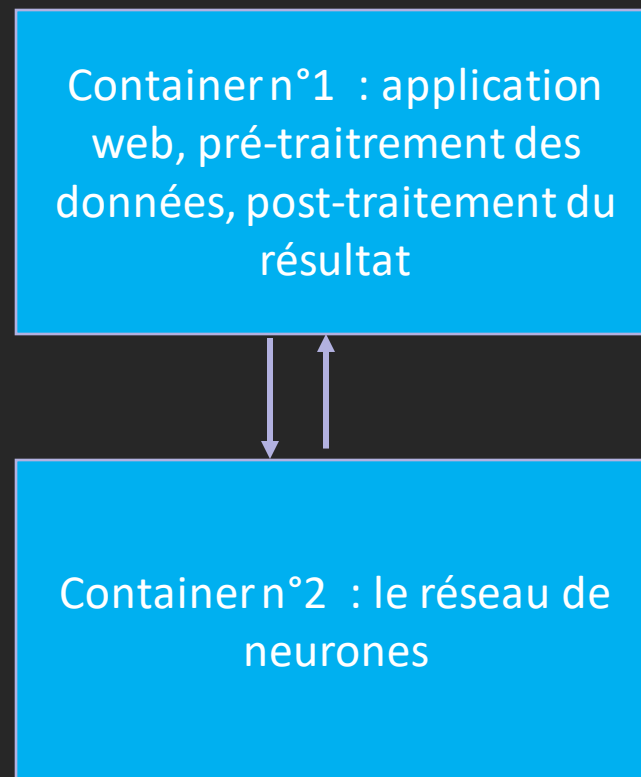
Flask permet de développer rapidement une [application web](#) rudimentaire pour permettre des premiers tests avec les utilisateurs

Vous pourrez facilement :

- Mettre en forme des pages web pour récolter des données ou présenter des résultats
 - Associer des fonctions python à ces pages web
-
- [Présentation rapide](#)

Solution 1 : serving

- Utiliser des solutions comme TensorFlow Serving ou MLFlow pour avoir le réseau de neurones dans un container distinct, accessible via une API REST
- Simplifie les opérations de mise à jour du réseau de neurones
- Coûteux car chaque requête passe par le réseau



Solution 2 : modèle local

- Réduit le nombre de conteneurs et donc les besoins en communications
- Nécessite de prendre en charge les mises jours
- Nécessite de configurer plus finement la sécurité et l'occupation mémoire
- Nécessite une solution pour paralléliser les demandes (RabbitMQ par exemple)

Container :

- Réception des données via appli web
- Pré-traitement
- Utilisation du réseau
- Post-traitement

Conclusion

Votez pour l'image qui représente le mieux votre état d'esprit.



Un feedback ?

Si vous souhaitez nous donner votre avis sur le lab, scannez ce QR Code !



Ressources

- Notebook qui a inspiré ce lab : [https://github.com/kmkarakaya/Deep-Learning-Tutorials/blob/master/Training_a_Hugging_Face_causal_language_model_from_scratch_\(TensorFlow\).ipynb](https://github.com/kmkarakaya/Deep-Learning-Tutorials/blob/master/Training_a_Hugging_Face_causal_language_model_from_scratch_(TensorFlow).ipynb)
- Article medium sur tous les concepts de GPT 2 (architecture) : <https://rowlando13.medium.com/everything-gpt-2-1-architecture-overview-132d16fe985a>
- Documentation Hugging Face sur les différents modèles GPT2 : https://huggingface.co/docs/transformers/model_doc/gpt2