# Modélisation d'Aptamères d'ADN IFT 3710

Dereck Piché, Jonas Gabirot, Guillermo Martinez

Avril 2023

Université de Montréal

# Contexte

# Contexte

Les aptamères (du latin *aptus* signifiant attaché) sont des séquences synthétiques d'ADN qui s'attachent à des molécules cibles soit pour les détecter, soit pour les neutraliser. Leur spécifité leur permet de s'attacher seulement à la cible voulue sans affecter leur environment.

# Contexte

L'énergie libre de Gibbs est une propriété des molécules qui représentent la capacité à faire du travail. Dans ce contexte, le travail à performer est l'attachement de l'aptamère avec la molécule cible.

Un aptamère doit avoir une énergie libre appropriée pour s'attacher à la molécule cible. C'est une condition nécessaire mais non suffisante.

# Objectif

Cependant, les molécules d'ADN sont assez larges et le calcul de l'énergie libre devient très complexe lorsque la taille des molécules augmentent. Notre objectif est donc d'entraîner un modèle capable de prédire l'énergie libre d'une séquence d'ADN quelconque, afin d'accélérer la recherche d'aptamère.

# Baseline: MLP

# MLP

Comme baseline, nous avons choisi un réseau MLP standard. Les MLPs ne font aucune supposition sur la structure de la séquence, ce qui permet de comparer avec d'autres modèles qui se basent sur des hypothèses différentes. Ils sont aussi faciles à implémenter et entraîner.

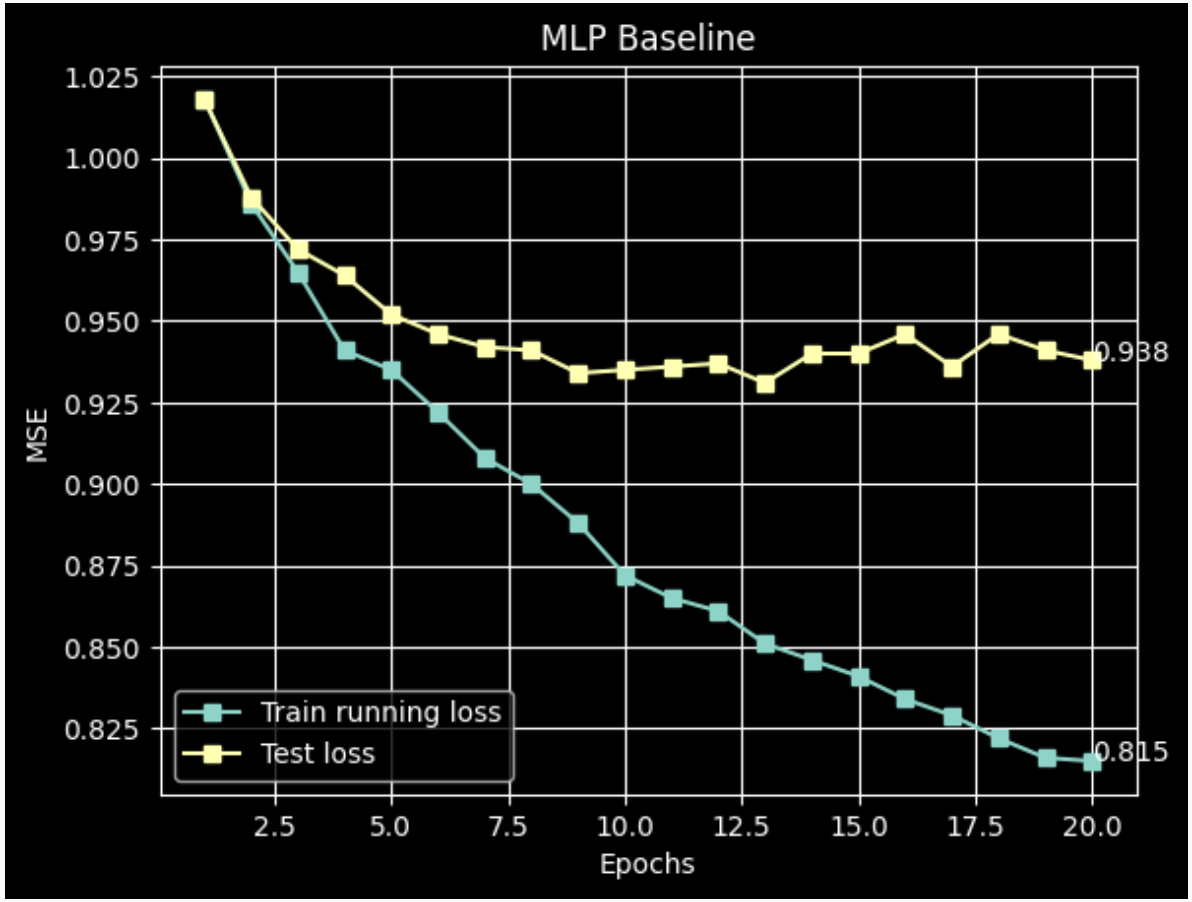
$$Seq \in R^{120} \xrightarrow{Lin.(\times 20)} \xrightarrow{ReLU(\times 20)} \xrightarrow{Lin.} Energie \in R$$

**Hyperparamètres**

1. Taux d'apprentissage: 0.0003
2. Nombre de paramètres: 1 005 241

# Résultats du MLP

**Figure 1:** Précision du MLP selon MSE



# Long Short Term Memory (LSTM)

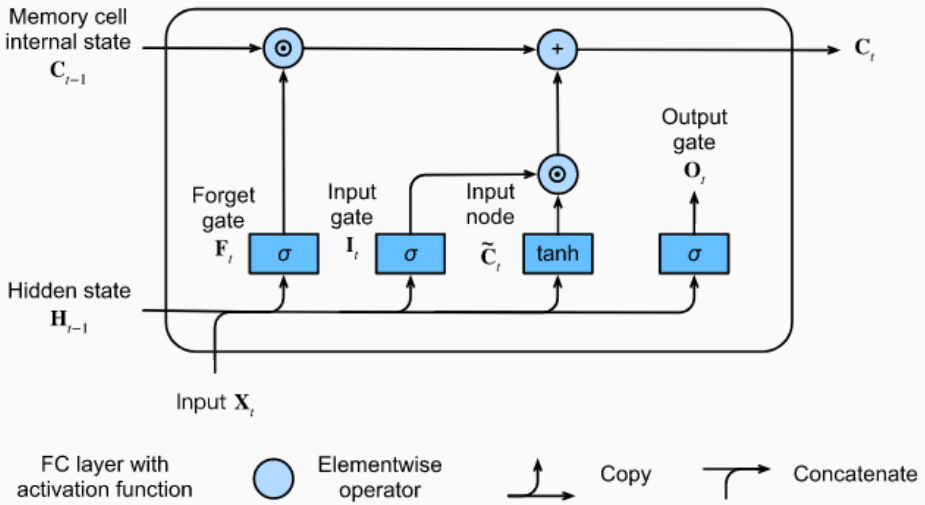
# LSTM

Le modèle LSTM (Long Short-Term Memory) est une variante des réseaux de neurones récurrents (RNN) qui permet de mieux gérer les problèmes liés à la mémoire à long terme. Il utilise des portes, qui sont des fonctions non linéaires, pour contrôler l'information qui entre ou sort de la mémoire.

# Portes

Il y a trois types de portes : la porte d'oubli (forget gate) permet au modèle de décider ce qu'il faut oublier de la mémoire à long terme, la porte d'entrée (input gate) permet d'ajouter de nouvelles informations à la mémoire à long terme et la porte de sortie (output gate) permet de déterminer quelle information doit être renvoyée en sortie.

**Figure 2:** Architecture LSTM



# Architecture du LSTM

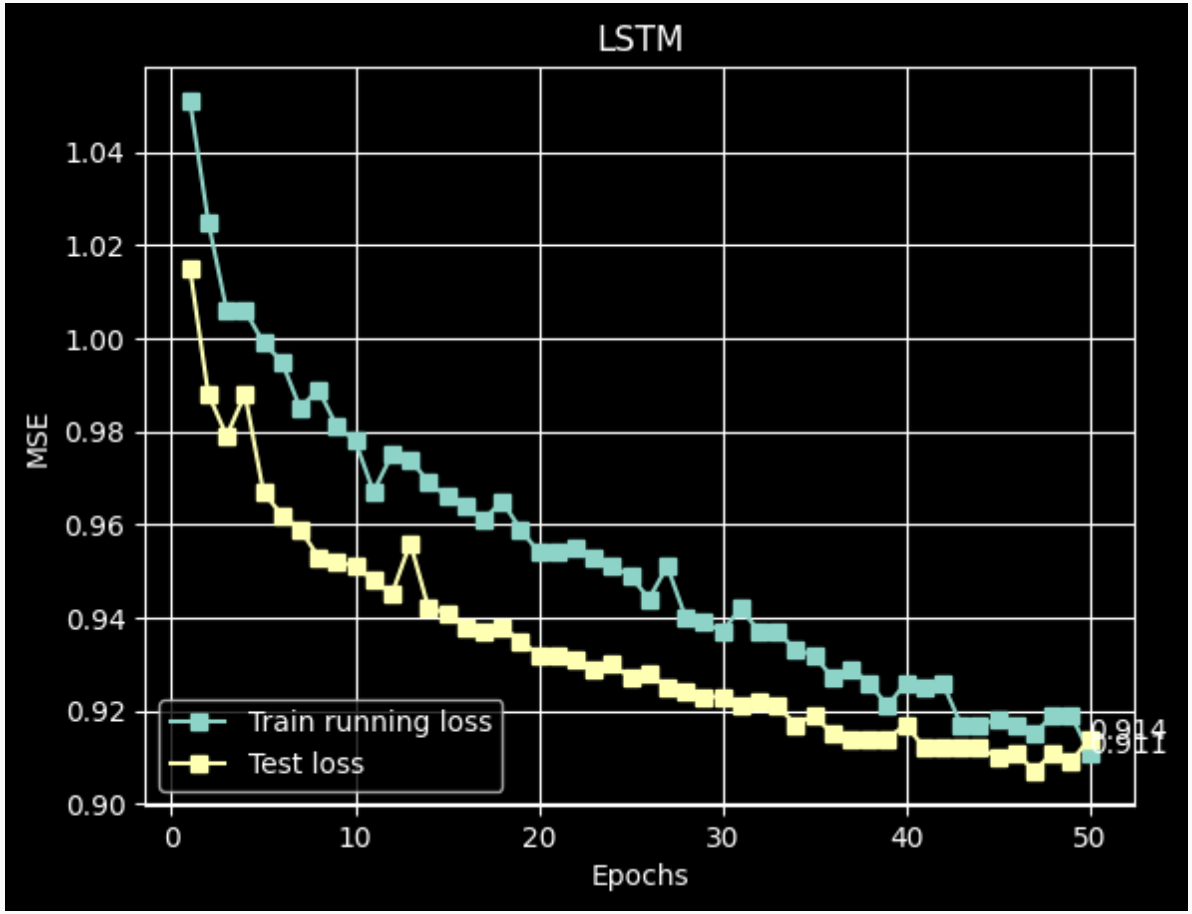
$$Seq \in R^{120} \xrightarrow{LSTM1} \xrightarrow{LSTM2} \xrightarrow{Lin.} Energie \in R$$

**Hyperparamètres**

1. Taux d'apprentissage: 0.0003
2. Nombre de paramètres: 1 008 991
3. Dropout: 0.6

# Résultats du LSTM

**Figure 3:** Précision du LSTM selon MSE



# Transformeurs

# Attention

Un système d'attention retourne une somme des vecteurs *Valeurs* selon une mesure de similarité entre les vecteurs *Requêtes* et les vecteurs *Clés*. Cette mesure de similarité est appelée l'attention.

**Système d'attention**

$\sum_j a(q, k_j) v_j$

Les transformeurs utilisent un système d'auto-attention, qui vise à produire une séquence transformée dans laquelle chaque jeton est codé en effectuant une somme pondérée des autres jetons présents dans sa séquence (l'incluant).

**Jeton codé**

$t_i' = \sum_j a(W^q t_i, W^k t_j) W^v t_j$

# Attention

Dans les transformeurs, on utilise une attention spéciale :

**Attention au produit scalaire mis à l'échelle**

$$a(q, k) = softmax(\frac{q^T k}{\sqrt{d}})$$

**Pourquoi le produit scalaire?** On sait déjà que le produit scalaire peut être vu comme une mesure de la similarité entre les deux vecteurs. Cela permet beaucoup de liberté au transformeur, qui obtient les vecteurs clés et les vecteurs requêtes en fonction des vecteurs jetons avant d'effectuer le produit scalaire. On le laisse donc créer sa propre mesure de distance, en quelque sorte.

**Pourquoi le softmax?** Pour que les coéfficients somment à 1. (*expliquer l'intuition)

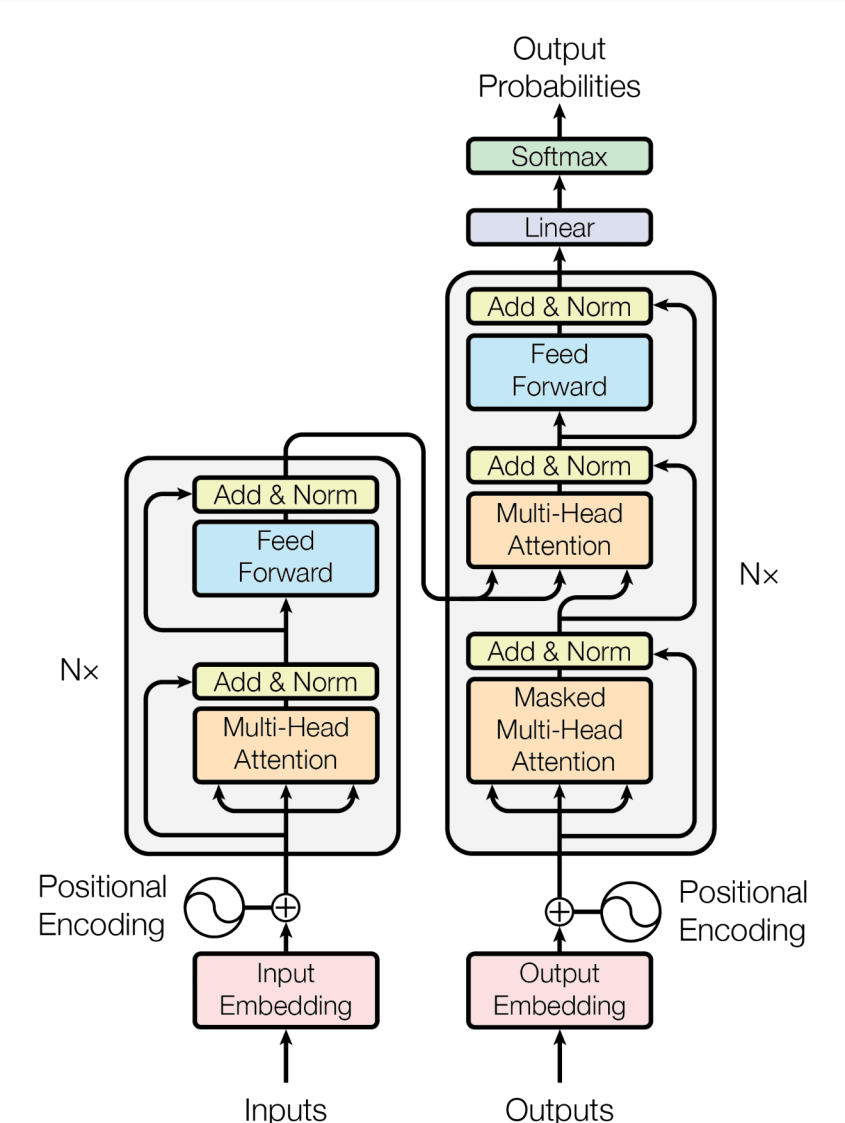
**Pourquoi la division par la racine de la taille?** Pour éviter des erreurs numériques. On épargne les détails.

# Couche de codage

Une tête d'attention retourne donc une séquence de la même taille que son entrée (sauf exceptions). Une couche de codage contient plusieurs têtes d'attention! Les sorties de ces têtes d'attention sont concaténées dans *un gros vecteur* et on applique une transformation linéaire (avec activation ReLU pour obtenir de l'expressivité non linéaire) à ce vecteur pour obtenir une sortie de dimension arbitraire.

# Architecture du transformeur

**Figure 4:** Transformer de traduction dans AIAYN



# Architecture du transformeur

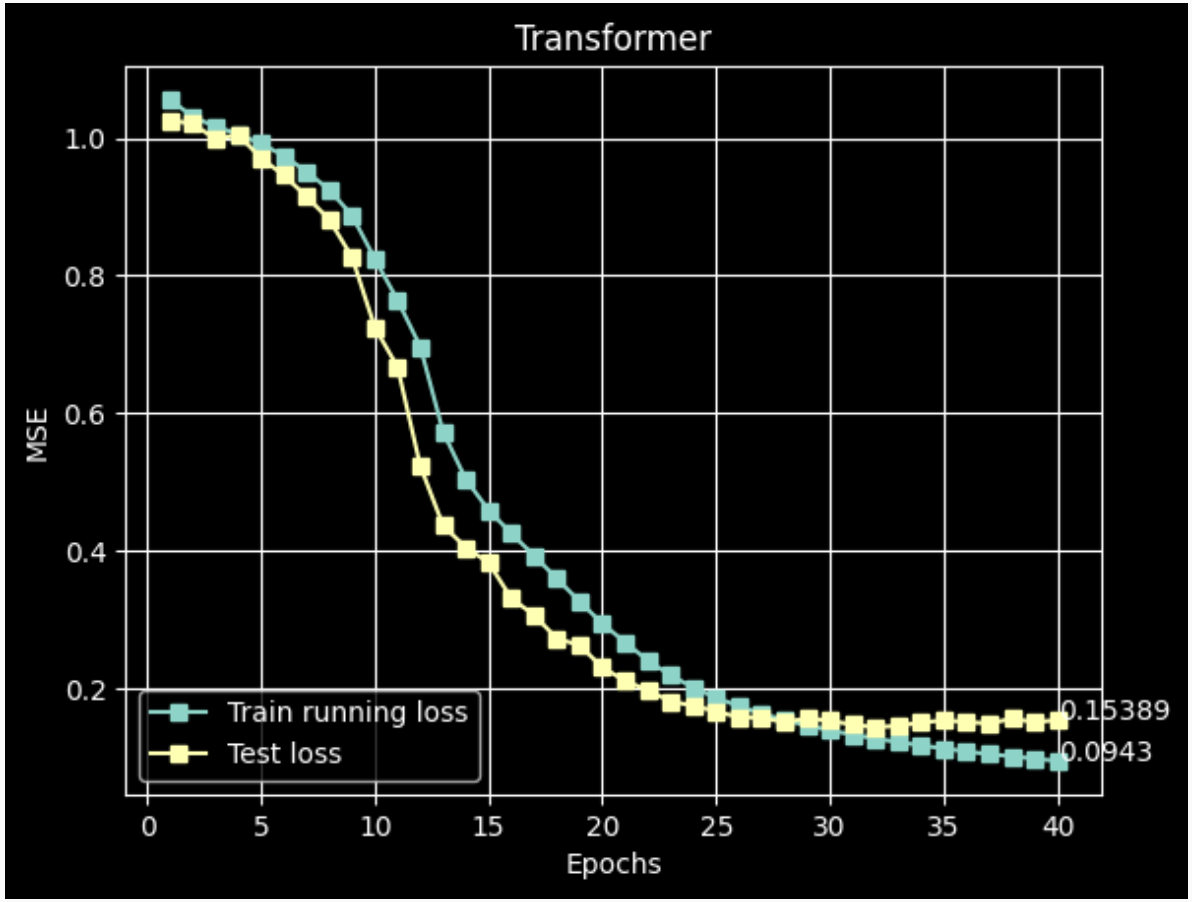
$$Seq \in R^{30} \xrightarrow{Emb.} \xrightarrow{Cod.(\times 3)} \xrightarrow{Lin.} Energie \in R$$

## Hyperparamètres

1. Taux d'apprentissage: $10^{-4}$
2. Dimension d'embedding: 16
3. Nombre de têtes: 8
4. Nombre de paramètres: 1 786 753
5. Codage positionnel: sincos (AIAYN)

# Résultats du transformeur

**Figure 5:** Précision du transformeur selon MSE



# Comparaison des résultats

**Figure 6:** MSE des 3 modèles