Les modèles de langue neuronaux

Projet de statistique appliquée

Etienne Boisseau, Olivier Dulcy, Christos Katsoulakis, Eric Lavergne,
sous la direction de Benjamin Muller (INRIA)

Mots-clés : *modèles de langue, n-gram, Transformer, Attention, perplexité*

Sommaire

Introduction

Notre projet de statistique appliquée a consisté en l'étude théorique et pratique des modèles de langue, en particulier ceux dits « neuronaux ».

Les modèles de langue sont des modèles qui portent sur le langage humain. L'objectif est de générer du texte ressemblant à ce qu'écrirait un humain. Pour cela, un modèle de langue apprend la structure de la langue souhaitée grâce à un texte qui est supposé représentatif de cette dernière. Il en retient alors la distribution probabiliste de la langue étudiée. Sachant un nombre de mots donnés en entrée, il est donc capable de prédire les mots qui pourraient suivre avec la plus forte probabilité. On peut penser à la suggestion de mots que propose un téléphone lorsqu'on écrit un message, qui relève de modèles de langue simples.

Lorsqu'ils sont suffisamment entraînés, ces modèles peuvent être utilisés pour de nombreuses tâches comme pouvoir résumer un texte ou être capable de répondre à des questions.

Récemment, les modèles de langue ont connu d'importantes avancées avec l'utilisation de réseaux de neurones profonds. Nous nous sommes penchés en particulier sur une architecture de réseaux de neurones appelée "Transformer", dont l'un des exemples phares est le modèle « GPT-2 » développé par OpenAI.

1 Les modèles de langue

1.1 Le modèle n-gram

Un des premiers modèles historiquement utilisé pour réaliser la tâche de génération de texte est le n-gram. Il se base sur l'idée intuitive que pour prédire le prochain mot d'un texte, les quelques derniers mots sont plus importants que les plus lointains. Formellement, le modèle n-gram fait l'hypothèse simplificatrice que la probabilité d'apparition d'un mot ne dépend que de ses $n-1$ prédécesseurs.

FIGURE 1 – Illustration du bi-gram : la probabilité d'apparition du dernier mot, ici « chat », est conditionné par 2 mots précédents

Cette approche a été étudiée et implémentée au début de notre projet pour avoir une meilleure compréhension des modèles de langue et générer des résultats de référence en vue de réaliser une comparaison avec un modèle plus complexe : le Transformer.

Le modèle n-gram, bien qu'intuitif et simple à mettre en oeuvre, souffre en effet de plusieurs inconvénients qui limitent la qualité des textes générés. Pour commencer, toute séquence de n mots n'apparaissant pas au moins une fois dans le corpus d'entrainement aura une probabilité nulle d'être générée. Si ce problème peut être partiellement résolu par certaines méthodes, la génération d'un texte trop différent du corpus d'entrainement restera de mauvaise qualité. La seule solution à ce problème est d'augmenter la quantité de données d'entraînement du n-gram exponentiellement avec n.

Un autre problème majeur des n-gram est la prise en compte d'un trop faible nombre de mots pour la prédiction. Le contexte global du texte n'est pas capturé et la génération peut rapidement devenir incohérente.

1.2 Le Transformer

Nous avons ensuite étudié et implémenté le Transformer, qui est actuellement une des formes les plus abouties de modèle de langue. Il s'agit d'une architecture de réseaux de neurones.

Pour que le modèle puisse traiter les entrées, il doit s'agir des nombres réels : il faut donc établir une réprésentation vectorielle des mots ou des fragments de mots. Un texte en entrée est ainsi d'abord traduit par le modèle en plusieurs vecteurs de nombres réels : c'est le *Word Embedding*, ou "plongement lexical" en français. Nous avons mis en oeuvre deux méthodes différentes de Word Embedding : à l'échelle des mots, c'est-à-dire de sorte à ce que chaque mot soit réprésenté par un vecteur différent, ainsi qu'à l'échelle des sous-mots (Subwords) (de l'ordre de la syllabe).

La sortie du modèle correspond aux probabilités estimées du mot suivant directement le texte en entrée. Pour entraîner le modèle, on lui donne un ou plusieurs mots d'un texte en entrée et on s'assure que le mot qui suit dans le texte ait une probabilité importante dans la sortie, en modifiant légèrement le modèle. On répète cela sur un nombre très important de phrases jusqu'à ce que le modèle soit performant.

Le modèle transforme successivement la représentation initiale des mots en entrée jusqu'à obtenir le vecteur final des probabilités du mot suivant.

L'attention

Un mécanisme central qui caractérise le Transformer est appelé l'"attention". Là où un modèle n-gram se contente de mémoriser toutes les séquences de n mots qu'il a déjà vues, le Transformer combine les représentations numériques des mots les unes avec les autres selon une fonction mathématique (l'attention) prévue pour mettre en relation les mots qui sont les plus pertinents les uns pour les autres. Par exemple, dans la phrase "Je suis un très beau chat", le mot "beau" sera mis en relation avec les mot "Je" et "chat", mais pas avec les mot "suis" et "un".

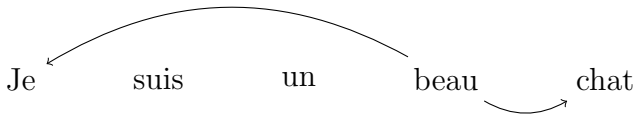

FIGURE 2 – Illustration du mécanisme de l'attention : « beau » se réfère à « chat » et « Je »

1.3 Méthodes de génération

Etant donné un modèle de langue entrainé, n-gram ou réseau de neurones, pour des considérations de ressources il n'est pas possible d'estimer une à une les probabilités de tous les textes possibles d'une certaine longueur puis de ne conserver que celui qui est le plus probable. Différentes stratégies plus fines sont possibles pour la génération de texte.

Une première méthode naïve, dite méthode gloutonne, consiste à constamment choisir pour mot suivant celui qui a la plus grande probabilité d'apparaître conditionnellement aux symboles précédents.

La méthode d'échantillonnage quant à elle consiste à choisir aléatoirement le mot suivant en donnant plus de poids aux mots les plus probables étant donné la séquence qui précède.

Enfin la méthode *Beam search*, consiste à garder en mémoire un ensemble de k échantillons pour finalement sélectionner le plus probable une fois arrivé à la longueur de texte voulue.

Parmi ces méthodes, la méthode d'échantillonnage est celle qui est le moins sensible à l'*overfitting* en évitant de générer systématiquement la même suite de symboles à partir d'un même contexte. En effet, elle permet l'exploration en générant des séquences plus diverses que les méthodes précédentes.

2 Expériences

2.1 Description

Afin de mettre en application ce que le reste de notre travail nous avait appris, nous avons implémenté en Python les algorithmes n-gram et Transformer.

Le jeu de données utilisé pour l'entraînement des modèles est un corpus de 1 million de paragraphes extraits du Wikipédia français. Pour une prise en main plus facile, le jeu de données est prétraité : tous les caractères sont en minuscules, et les mots et les signes de ponctuation sont séparés (en anglais, ce traitement s'appelle la *tokenization* d'un texte).

Deux versions de ces algorithmes ont été implémentées : une utilisant la librairie TensorFlow et la réprésentation Subwords, l'autre utilisant la librairie PyTorch et la représentation Word-Level. De cette manière nous avons pu observer les avantages et inconvénients de ces deux représentations, ainsi que les particularités de deux librairies très populaires en Deep Learning.

2.2 Résultats

Le tableau suivant récapitule les performances des modèles que nous avons entraînés selon la mesure d'erreur appelée *perplexité* (on désire une perplexité la plus petite possible).

Les colonnes distinguent la performance sur le jeu de données d'entraînement (*train*) et sur un jeu de données d'évaluation (*test*). Le fait de mesurer la performance sur un jeu de données non utilisé pour l'entraînement du modèle permet de s'assurer que sa performance ne soit pas dû à de la simple mémorisation du texte d'origine mais à une véritable modélisation.

Modèle	train	test
2-gram Subwords	378.72	381.17
Transformer Subwords	18.97	19.26
2-gram Word-Level	79.6	89.1
Transformer Word-Level	27.7	26.4

Table 1 – Perplexité des modèles testés. Un nombre faible indique une bonne performance.

Dans le cas Subwords et Word-Level, la perplexité est inférieure avec le Transformer qu'avec le 2-gram. Ceci confirme ce à quoi l'on pouvait s'attendre : le modèle Transformer est capable de modéliser le langage naturel bien mieux que le modèle naïf qu'est le n-gram.

Plus qualitativement, voici un exemple de texte généré par le modèle Transformer Subwords à partir de quelques mots de départ choisis à l'avance :

> "A l'age de 5 ans , elle invente" → "le premier tour de l' unite des etats-unis . elle est egalement connue pour ses elections a paris , et se retrouve dans le cinema en 1968 ... , qui a fait la connaissance d' un grand nombre de secondes et de residences d' argent , dont la ville est la premiere et la plus grande virginie

de son pere”

Les conclusions issues des comparaisons quantitatives sont également observables qualitativement. Les générations du transformer sont généralement mieux construites que celles du 2-gram, elles suivent davantage les règles syntaxiques et conservent un sens sur une plus longue durée.

Ce projet a également pu mettre en avant les avantages et inconvénients des schèmes de tokenization. Le fait de séparer le texte en mots est naturel (Word-Level), et permet au modèle d'apprendre vite. Cependant, le nombre total de mots dans la langue française étant très grand, nous sommes obligés de restreindre le vocabulaire du modèle et de remplacer les mots rares par le mot inconnu, ”<unk>”.

La séparation d'un texte en Subwords (groupes de quelques lettres) permet de résoudre ce problème, car le modèle ne rencontre plus de symboles inconnus. En revanche, chaque texte est séparé en un plus grand nombre de symboles et paraît plus long aux yeux du modèle. Ceci diminue la cohérence globale des textes générés.