Une méthodologie pour l'implémentation d'applications tierces intelligentes à destination des assistants vocaux via des bandits linéaires

Robin Allesiardo*, Christophe Sauldubois*, Fabrice Depaulis[1] **, Nicolas Bulteau[1]**, Frédéric Chantrel[1]**, Erwan Pigneul *

* Solocal,
rallesiardo/epigneul/csauldubois@solocal.com
** Orange,
prenom.nom@orange.com

Résumé. Les applications tierces déployées sur les assistants vocaux (Google Home, Amazon Echo) suivent habituellement des règles de gestions basées sur un graphe de dialogue codé en dur. Dans ce papier, nous décrivons la manière dont nous avons inclu de l'intelligence artificielle dans notre application, à destination d'Amazon Echo et de la Google Home, déployée actuellement en production. Notre approche est basée sur un algorithme de bandits contextuels qui permet de piloter le dialogue à l'intérieur d'un graphe de dialogue flou tout en utilisant les fonctionnalités et les variables mises à disposition par les frameworks propriétaires utilisées pour la création d'applications tierces.

1 Introduction

Les agents conversationnels se sont démocratisés au travers des smartphones (Google Assistant, Siri) et des assistants vocaux (Google Home, Amazon Echo...). Les utilisateurs peuvent installer des applications tierces qui interagissent avec ces périphériques via des briques logicielles propriétaires mises à la disposition des développeurs. Notre cas d'utilisation est la recherche de professionnels au travers des assistants vocaux (par exemple, un plombier ou un restaurant) et entre dans la famille des systèmes de dialogues orientés tâches. Ces systèmes sont conçus pour remplir une tâche particulière, en interagissant avec un utilisateur et n'ont pas pour but de participer à des conversations non structurés sans rapport avec les tâches à accomplir. Dans ce papier, nous décrivons les méthodes et algorithmes utilisées par notre système de dialogue orienté tâches et comment les intégrer au sein de l'éco-système technique associé aux assistants vocaux pour ajouter de l'intelligence aux application tierces. En effet, même si des algorithmes de traitement du langage naturel sont utilisés au sein des assistants vocaux, les applications tierces sont contraintes de suivre un graphe de dialogue prédéfini pour associer les requêtes utilisateurs aux fonctionnalités des applications. Notre approche tire partie

1. Travaux effectués au sein de Solocal

d'une fonctionnalité particulière des assistants vocaux pour transférer le contrôle de la conversation de l'assistant vocal vers notre système de dialogue personnalisé afin de nous permettre d'entrainer notre modèle de conversation dynamiquement au contact des utilisateurs.

État de l'art

Les applications tierces sont crées à partir de bibliothèques propriétaires. Certaines d'entre-elles permettent de concevoir des applications portables sur plusieurs plateformes. Pour la plupart, ces bibliothèques partagent les concepts d'intentions et d'entités.

Intentions : la phrase parlée de l'utilisateur n'est pas toujours disponible au niveau application. À la place, le périphérique analyse l'entrée audio et renvoie une intention à l'application. Une intention est une correspondance entre une requête utilisateur et un ensemble d'actions utilisateurs définies par l'application. Par exemple, la phrase "Je cherche un restaurant" peut être associée à l'intention "recherche_restaurant". Sur Amazon Echo, ni l'audio, ni sa transcription ne sont transmis à l'utilisateur, uniquement l'intention détectée.

Entités : ce sont les valeurs des paramètres associés à chaque intention et sont extraites des requêtes utilisateurs. Par exemple, dans "je cherche un restaurant à Londres", le mot "Londres" pourrait être extrait comme une entité "localisation".

Les applications tierces sont supposées être dirigées par un graphe de dialogue statique contrôlé par les différentes intentions et entités détectées durant le dialogue. Le système doit connaitre les différentes transitions entre l'intégralité des état de dialogue possible.

L'apprentissage par renforcement a été étudié pour l'entrainement de systèmes de dialogue. Dans RLDS [10], le dialogue est considéré comme un processus de décision Markovien et les états du dialogue sont estimés grâce à un oracle. Les travaux récents visent à entrainer les systèmes de dialogue par l'apprentissage par renforcement combiné à des réseaux de neurones profonds [4]. Cependant, ces algorithmes nécessitent de grosses quantités de données et ne peuvent pas être déployé en production sans un entrainement préalable. Pour résoudre ce problème, une approche utilisant un magicien d'Oz [9] a été proposée. Celle-ci est cependant difficile à utiliser en pratique car un opérateur humain doit répondre à la place de l'agent pendant une longue période de temps. Une seconde approche utilise des bots utilisateurs [7, 8], codés en dur, ayant chacun un comportement reflétant un cas d'utilisation. Cette approche permet d'entrainer l'agent aux contacts des bots, remplaçant le besoin d'utilisateurs. Ces deux méthodes ne sont pas utilisables dans notre cas car elles nécessitent la requête textuelle, qui n'est pas disponible sur notre cible, Amazon Echo.

2 Le système de dialogue

Nous définissons un dialogue comme une séquence de tours de dialogue entre un utilisateur et le système. Après chaque entrée utilisateur, le système renvoie une réponse. Cette réponse peut être parlée ou bien être une action effectuée par le périphérique. Les kits de développement Alexa et DialogFlow sont basés sur la même architecture. Après une requête utilisateur, le périphérique lui associe une intention et en extrait des entités. Ensuite, la réponse correspondante au prochain nœud du graphe de dialogue est retournée. Afin de gagner en flexibilité, les développeurs tiers peuvent utiliser un webhook. Un webhook permet d'envoyer des informations à une API au travers d'une requête POST pour requêter une base de données ou mettre à jour

des informations utilisateur. Nous avons cependant utilisé le webhook pour court-circuiter le graphe de dialogue du périphérique en routant toutes les intentions vers le même appel d'API. Les intentions et entités sont traitées dans notre système de dialogue personnalisé et la réponse textuelle est ensuite envoyée au périphérique vocal pour être lue à l'utilisateur.

2.1 Le système de décision

Notations. Soient $i \in \mathbb{N}^+$ un index de conversation et $t \in \mathbb{N}^+$ le nombre de tours de conversation depuis le début du dialogue i. Soit $k \in [K]$ une action et $[K]$ l'ensemble des actions à la disposition du système. Soit D_k la dimension du contexte $x_{i,t,k} \in \mathbb{R}^{D_k}$ associé à l'action k. Étant donné les contextes actuels, $[K]_{i,t} \subseteq [K]$ est l'ensemble des actions disponibles au tour t de la conversation i. Après avoir joué l'action $k_{i,t}$, une récompense $r_{i,t} \in [0, 1]$ est générée.

L'algorithme de bandits linéaires Un dialogue est un problème à information partielle avec un compromis entre l'exploration et l'exploitation. En effet, ne connaissant pas la politique de dialogue optimale à son initialisation, il est nécessaire pour l'algorithme d'explorer afin de trouver les actions permettant ensuite de maximiser sa récompense cumulée (la somme des $r_{i,t}$). Ce problème est connu comme le problème des bandits contextuels. LinUCB [6, 3] est un algorithme de bandits contextuels qui utilise des modèles linéaires pour prédire les récompenses des actions tout en maintenant un intervalle de confiance supérieur sur les récompenses pour engendrer une exploration optimiste des actions. Bien qu'étant linéaire, LinUCB possède de fortes garanties théoriques et peut-être compétitif sur le court terme face à des algorithmes basés sur des réseaux de neurones [1] ou des forêts aléatoires [5] grâce à sa rapidité de convergence.

La fonction de récompense utilise un retour en N-étapes, inspiré de l'apprentissage TD [11], pour permettre au modèle d'apprendre des dépendances temporelles. Cela permet au modèle d'être récompensé avec des récompenses obtenues plusieurs itérations après avoir joué une action et un facteur de "discount" est utilisé de manière à favoriser les récompenses immédiates plutôt que celles sur le long terme. Ceci permet d'éliminer les actions inutiles n'ayant pas d'impact positif sur les performances.

Soit $\gamma \in [0, 1]$ un facteur de "discount". La récompense $r_{i,t}$ est générée après avoir joué l'action $k_{i,t}$. Chaque action $k_{i,t'} \in H_i$est récompensée avec la récompense $r_{i,t}\gamma^{t-t'}$. Au temps T, la récompense cumulée de l'action $k_{i,t}$, avec $t \leq T$ est :

$$y_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} + ... + \gamma^{T-t} r_{i,T}$$

Nous utilisons une version modifiée de LinUCB (voir Algorithme 1) où chaque action peut avoir des contextes de tailles différentes. La linéarité de LinUCB permet aussi de mettre à jour les modèles immédiatement lorsqu'une récompense intermédiaire est disponible.

2.2 Le gestionnaire de dialogue

Deux objets principaux sont manipulés, les contextes et les actions. Les actions sont des réponses possibles à une requête utilisateur. Par exemple "Je vous propose param :POI à param :localisation". Des appels à des systèmes externes peuvent aussi être associés aux actions, comme une réservation de table ou l'achat de billets d'avions. Les contextes sont construits par le système et reflètent l'état de la conversation. Les variables principales des contextes sont les intentions et les entités. Bien que les intentions et les entités soient créés manuellement, des

Algorithme 1 LinUCB Asynchrone avec relations temporelles

// Initialisation des modèles linéaires
Entrées : $\alpha \in \mathbb{R}^+, \gamma \in [0, 1]$
for $k \in [K]$ **do**
  $A_k \leftarrow I_{D_k}$ // la matrice identité de dimensions D_k par D_k
  $b_k \leftarrow 0_{D_k}$
end for
for each call do
  for $k \in [K]_{i,t}$ **do**
    $x_k \leftarrow x_{i,t,k}$
    $\theta_k \leftarrow A^{-1}b$
    $p_k \leftarrow \theta_k^T x_k + \alpha\sqrt{x_k^T A^{-1} x_k}$ /* l'intervalle de confiance est ajouté à la récompense */
  end for
  $k_{i,t} \leftarrow \arg\max_{k \in [K]_{i,t}} p_k$
  $A \leftarrow A + x_k x_k^T$
  Observer la récompense $r_{i,t}$
  for $j \in \{0, .., t\}$ **do**
    $b_{k_{i,j}} \leftarrow b_{k_{i,j}} + x_{i,j,k_{i,j}} \gamma^{t-j} r_{i,t}$
  end for
end for

méthodes existent pour les créer de manière automatique depuis des historiques de dialogues existants [2]. Afin d'aider l'algorithme d'apprentissage, plusieurs méthodes permettent de le guider et de réduire l'espace d'exploration.

Des intentions et des actions génériques : une intention "recherche_restaurant" peut être remplacée par "recherche_professionnel" ayant pour paramètre "param :professionnel". Procéder de la sorte permet de déléguer l'analyse générale de la requête au périphérique et d'analyser nous même les entités détectées. Définir les bonnes intentions a été particulièrement important car notre cible principale, Amazon Alexa, ne restitue pas la requête utilisateur, mais uniquement le nom de l'intention et les valeurs des entités.

Contraindre le graphe avec des déclencheurs : pour guider les choix de l'algorithme et s'assurer de sa sureté, nous avons ajouté des déclencheurs (règles métiers) contrôlant les actions disponibles dans $[K]_{i,t}$. Certaines règles peuvent forcer l'algorithme à choisir une action particulière (avec un ensemble d'action de taille 1) tandis-que d'autres peuvent simplement retirer une action de l'ensemble. Ceci est important pour permettre de protéger certains actions derrière une action de confirmation.

Construire de contexte de dialogue : chaque système de dialogue peut contextualiser des variables différentes, dépendantes des tâches à accomplir, et chaque action peut avoir des contextes de tailles différentes. Quelques exemples de variables : le nombre de tours de dialogue, l'intention détectée, la liste des actions précédentes, le nombre de résultats de recherches (utile pour les actions de filtrage). Ces variables sont regroupées avant chaque décision pour former les vecteurs de contextes. Il est important de préciser que LinUCB est robuste aux contextes générés par un adversaire. Cela signifie que l'algorithme peut-être utilisé même lorsque les contextes ne sont pas tirés depuis des distributions stationnaires.

Les récompenses sont définie manuellement et sont déclenchées lorsque l'utilisateur effectue une action entrainant une conversion (déclenchement d'un appel téléphonique, d'un achat, d'une réservation). Les nœuds du dialogue n'entrainant pas de récompense seront peu à peu négligés puis éliminés au profit de chemins aux récompenses plus élevées.

2.3 Les utilisations pratiques du système de décision

Bien que cette approche ouvre de nombreuses possibilités, cette architecture n'est pas supposée fournir un dialogue contrôlé par une IA de bout en bout comme l'envisagent les méthodes d'apprentissage profond. Elle peut être utilisé principalement :

L'optimisation de la formulation. La formulation des réponses peut impacter l'expérience utilisateur. Même si la sémantique des deux phrases parlées est identique, une mauvaise formulation peut porter à confusion tandis-qu'une bonne formulation pour augmenter le taux de complétion des tâches.

Le remplacement des règles expertes. Ce type de système de dialogue comprend un certain nombre de règles expertes. Certaines sont factuelles, comme la protection d'une action de réservation par une action de confirmation, et peuvent raisonnablement être définies manuellement. Cependant d'autre règles relèvent de l'heuristique et bénéficient grandement d'un apprentissage dynamique. Par exemple, que faire si l'utilisateur n'a pas précisé la localisation de sa recherche ? Est-ce qu'il faut utiliser la géo-localisation du périphérique ? Ou bien la ville entrée dans son profil ? La localisation de la dernière recherche ? Qu'en est-il si cette recherche a été effectuée il y a 30 secondes, ou bien 2 jours ? Le système de décision peut résoudre ce type de problème. Après avoir ajouté ces informations au vecteur de contexte, la règle sera apprise de manière automatique par le système de décision.

3 Conclusion

Dans ce papier, nous avons décrit les méthodes et algorithmes utilisés par notre système de dialogue orienté tâches déployé dans notre environnement de production. Nous avons proposé une nouvelle approche permettant d'intégrer des méthodes d'apprentissage par renforcement de bandits contextuels dans des assistants vocaux tels que Amazon Echo ou Google Home pour ajouter de l'intelligence aux application tierces. Le développement des applications pour les assistants vocaux est toujours à ses débuts, sans conventions, et ceci est à notre connaissance la première description de la mise en pratique d'algorithmes d'apprentissage par renforcement dans une application tierce. Nous pensons que cette architecture est une manière élégante de faire cohabiter des règles expertes ainsi que de l'intelligence artificielle dans les assistants vocaux.

Références

[1] Robin Allesiardo, Raphaël Féraud, and Djallel Bouneffouf. A neural networks committee for the contextual bandit problem. In *Neural Information Processing - 21st International Conference, ICONIP*, pages 374–381, 2014.

[2] Jean-Léon Bouraoui and Vincent Lemaire. Cluster-based graphs for conceiving dialog systems. In *Workshop DMNLP at European Conference on Machine Learning (ECML)*, 2017.

[3] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 208–214, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[4] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. End-to-end reinforcement learning of dialogue agents for information access. Technical report, September 2016.

[5] Raphaël Féraud, Robin Allesiardo, Tanguy Urvoy, and Fabrice Clérot. Random forest for the contextual bandit problem. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 93–101, 2016.

[6] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM.

[7] Xiujun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. End-to-end task-completion neural dialogue systems. *CoRR*, abs/1703.01008, 2017.

[8] Xiujun Li, Zachary C. Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. A user simulator for task-completion dialogues. *CoRR*, abs/1612.05688, 2016.

[9] Lina Maria Rojas-Barahona, Milica Gasic, Nikola Mrksic, Pei-Hao Su, Stefan Ultes, Tsung-Hsien Wen, Steve J. Young, and David Vandyke. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1 : Long Papers*, pages 438–449, 2017.

[10] Satinder P. Singh, Michael J. Kearns, Diane J. Litman, and Marilyn A. Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 956–962, 1999.

[11] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 :9–44, 1988.

Summary

Third-party applications deployed on vocal home-devices (Google Home, Amazon Echo...) are usually rule-based and follow an hard-coded dialogue graph. In this paper we describe how we included artificial intelligence in our vocal conversational agent actually running in production on Amazon Echo and soon on Google Home. This approach is based on contextual bandits, a special case of reinforcement learning, that allows to pilot the dialogue inside a fussy dialogue graph while taking advantage of the features available in the home-devices' frameworks.