Université de Montréal

Prédiction et génération de données structurées à l'aide de réseaux de neurones et de décisions discrètes

par

Francis Dutil

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

août 2018

Sommaire

L'apprentissage profond, une sous-discipline de l'apprentissage automatique, est de plus en plus utilisé dans une multitude de domaines, dont le traitement du langage naturel. Toutefois, plusieurs problèmes restent ouverts, notamment la prédiction de longues séquences et la génération de langues naturelles. Dans le mémoire qui suit, nous présentons deux modèles travaillant sur ces problèmes.

Dans le chapitre 1, nous incorporons un système de planification à l'intérieur des modèles séquence-à-séquence. Pour ce faire, le modèle détermine à l'avance l'alignement entre la séquence d'entrée et de sortie. Nous montrons que ce mécanisme améliore l'alignement à l'intérieur des modèles, converge plus rapidement et nécessite moins de paramètres. Nous montrons également des gains de performance en traduction automatique, en génération de questions ainsi que la découverte de circuits eulériens dans des graphes.

Dans le chapitre 2, nous appliquons des réseaux antagonistes génératifs aux langues naturelles, une tâche compliquée par la nature discrète du domaine. Le modèle est entraîné de manière purement non supervisée et n'utilise aucune estimation de gradients. Nous montrons des résultats en modélisation de la langue, en génération de grammaires non contextuelles et génération conditionnelle de phrases.

Mots clés : intelligence artificielle, apprentissage automatique, apprentissage profond, réseaux de neurones récurrents, réseaux de neurones à convolution, réseaux antagonistes génératifs, décisions discrètes

Summary

Deep learning, a subdiscipline of machine learning, is used throughout multiple domains, including natural language processing. However, in the field multiple problems remain open, notably the prediction of long sequences and the generation of natural languages. In the following thesis, we present two models that work toward solving both of these problems.

In chapter 1, we add a planning mechanism to sequence-to-sequence models. The mechanism consists of establishing ahead of time the alignment between the input and output sequence. We show that this improves the alignment, help the model to converge faster, and necessitate fewer parameters. We also show performance gain in neural machine translation, questions generation, and the algorithmic task of finding Eulerian circuits in graphs.

In chapter 2, we tackle the language generation task using generative adversarial networks. A non-trivial problem considering the discrete nature of the output space. The model is trained using only an adversarial loss and without any gradient estimation. We show results on language modeling, context-free grammar generation, and conditional sentence generation.

keywords: artificial intelligence, machine learning, deep learning, recurrent neural networks, convolutional neural networks, generative adversarial networks, discrete decisions

Liste des sigles et des abréviations

BLEU	Étude d'évaluation bilingue, de l'Anglais *BiLingual Evaluation Understudy*
CNN	Réseau de neurones à convolution, de l'Anglais *Convolutional Neural Network*
GAN	Modèles génératifs adversariaux, de l'Anglais *Generative Adversarial Netoworks*
GRU	Réseau récurrent à portes, de l'Anglais *Gated Recurrent Network*
i.i.d	Variables indépendantes et identiquement distribuées, de l'Anglais *Independent and Identically Distributed*
LSTM	Réseau récurrent à mémoire court et long terme, de l'Anglais *Long short-term memory*
MLE	Estimateur de maximum de vraisemblance, de l'Anglais *Maximum Likelyhood Estimator*
MLP	Perceptron multi-couches, de l'Anglais *Multi Layer Perceptron*
MSE	Erreur quadratique moyenne, de l'Anglais *Mean Square Error*
NLL	Log vraisamblence négative, de l'Anglais *Negative Log Likelihood*
NMT	Traduction automatique neuronale, de l'Anglais *Neural Machine Translation*
reLU	Unité linéaire rectifiée, de l'Anglais *Rectified Linear Unit*
RNN	Réseau de neurones récurrent, de l'Anglais *Recurrent Neural Network*
SGD	Descente de gradient stochastique, de l'Anglais *Stochastic Gradient Descent*

Table des matières

Sommaire iii

Summary v

Liste des sigles et des abréviations vii

Liste des tableaux xiii

Table des figures xv

Remerciements xvii

Introduction 1

0.1. Apprentissage automatique 1

0.1.1. Introduction 1

0.1.2. Type d'apprentissage automatique 2

0.1.3. Fonction de coût 3

0.2. Réseaux de neurones 4

0.2.1. Fonction d'activation 6

0.2.2. Entraînement 7

0.2.3. Rétropropagation du gradient 9

0.2.4. Normalisation dans les réseaux de neurones 11

0.3. Données séquentielles 12

0.3.1. Réseaux de neurones récurrents 13

0.3.1.1. Teacher forcing 15

0.3.1.2. Rétropropagation à travers le temps 16

0.3.1.3. Dépendance à long terme 16
0.3.1.4. Réseau récurrent à mémoire court et long terme 18
0.3.1.5. Gated Recurent Units 19
0.3.1.6. Normalisation dans les réseaux récurrents 20
0.3.2. Réseaux de neurones à convolution 21
0.4. Modèles séquence-à-séquence 23
0.4.1. Séquence-à-séquence avec attention 23
0.4.2. Application 25
0.5. Modèles génératifs 26
0.5.1. Réseaux Antagonistes Génératifs 26
0.6. Décision discrète 28
0.6.1. REINFORCE 29
0.6.2. Straight-Through Estimator 30
0.6.3. Gumbel-softmax 30
Chapitre 1. Planification dans les modèles séquence-à-séquence 31
1.1. Abstract 32
1.2. Introduction 32
1.3. Related Works 34
1.4. Planning for Sequence-to-Sequence Learning 35
1.4.1. Notation and Encoder 35
1.4.2. Alignment and Decoder 35
1.4.2.1. Alignment Repeat 38
1.4.3. Training 39
1.5. Experiments 40
1.5.1. Algorithmic Task 41
1.5.2. Question Generation 41

1.5.3. Character-level Neural Machine Translation 42

1.6. Conclusion 44

Chapitre 2. Génération de langue naturelle à l'aide de modèles génératifs. 45

2.1. Abstract 46

2.2. Introduction 46

2.3. Generative Adversarial Networks 48

2.4. Model architecture 49

2.4.1. Recurrent Models 49

2.4.2. Convolutional Models 51

2.4.3. Curriculum Learning 51

2.5. Experiments & Data 52

2.5.1. Simple CFG 53

2.5.2. Penn Treebank PCFG 53

2.5.3. Chinese Poetry 53

2.5.4. Language Generation 54

2.5.5. Conditional Generation of Sequences 54

2.5.6. Training 55

2.6. Results and Discussion 55

2.7. Conclusion and Future work 57

Chapitre 3. Conclusion 59

Bibliographie 61

Liste des tableaux

1.1 The results of different models on the WMT'15 tasks for English to German, English to Czech, and English to Finnish language pairs. We report BLEU scores of each model computed via the *multi-blue.perl* script. The best-score of each model for each language pair appears in bold-face. We use *newstest2013* as our development set, *newstest2014* as our "Test 2014" and *newstest2015* as our "Test 2015" set. $(^{\dagger})$ denotes the results of the baseline that we trained using the hyperparameters reported in [**Chung *et al.*, 2016**] and the code provided with that paper. For our baseline, we only report the median result, and do not have multiple runs of our models. On WMT'14 and WMT'15 for En*rightarrow*De character-level NMT, [**Kalchbrenner *et al.*, 2016**] have reported better results with deeper auto-regressive convolutional models (Bytenets), 23.75 and 26.26 respectively. 43

2.1 Simple CFG task results 56

2.2 BLEU scores on the poem-5 and poem-7 datasets 57

2.3 Word and character-level generations on the 1-billion word dataset 57

2.4 Word level generations on the Penn Treebank and CMU-SE datasets 58

2.5 Conditional generation of text. Top row shows generated samples conditionally trained on amazon review polarity dataset with two attributes 'positive' and 'negative'. Bottom row has samples conditioned on the 'question' attribute 58

Table des figures

0.1 Représentation graphique des réseaux de neurones avec une couche cachée. Figure tirée de [**Larochelle, 2013**]. 6

0.2 Représentation graphique des réseaux de neurones multicouche (MLP) 7

0.3 Descente de gradients dans l'espace des paramètres. Les paramètres sont mis à jour selon la fonction de coût. Figure tirée de [**Larochelle, 2013**]. 8

0.4 Un exemple mettant en évidence le problème d'utiliser les perceptrons multicouches sur des données séquentielles. Le modèle doit trouver l'information essentielle, peu importe son emplacement. Une redondance est donc nécessaire entre w_2 et w_4 . . . 13

0.5 Différentes configurations de réseau de neurones récurrents 15

0.6 Comparaison entre la cellule de récurrence du LSTM et du GRU. Les différentes portes sont mises en évidence. Figure tirée de [**Chung *et al.*, 2014**]. 19

0.7 Différente configuration de réseaux de neurones à convolutions 22

0.8 Comparaison des modèles Encodeur-Décodeur avec et sans attention. 24

0.9 Alignement des mots entre les phrases sources et cible. Figure tirée de [**Bahdanau *et al.*, 2014a**]. 25

0.10 Exemple de GAN. Le générateur produit des images de chats et le discrimateur détermine si elles sont réalistes ou non. 27

1.1 Our planning mechanism in a sequence-to-sequence model that learns to plan and execute alignments. Distinct from a standard sequence-to-sequence model with attention, rather than using a simple MLP to predict alignments our model makes a plan of future alignments using its alignment-plan matrix and decides when to follow the plan by learning a separate commitment vector. We illustrate the model

for a decoder with two layers $\mathbf{s}'_t$ for the first layer and the $\mathbf{s}_t$ for the second layer of the decoder. The planning mechanism is conditioned on the first layer of the decoder ($\mathbf{s}'_t$). 37

1.2 We visualize the alignments learned by PAG in (a), rPAG in (b), and our baseline model with a 2-layer GRU decoder using $\mathbf{s}_2$ for the attention in (c). As depicted, the alignments learned by PAG and rPAG are smoother than those of the baseline. The baseline tends to put too much attention on the last token of the sequence, defaulting to this empty location in alternation with more relevant locations. Our model, however, places higher weight on the last token usually when no other good alignments exist. We observe that rPAG tends to generate less monotonic alignments in general. 40

1.3 Learning curves for question-generation models on our development set. Both models have the same capacity and are trained with the same hyperparameters. PAG converges faster than the baseline with better stability. 42

1.4 Learning curves for different models on WMT'15 for En→De. Models with the planning mechanism converge faster than our baseline (which has larger capacity). 44

2.1 Model architecture 49

2.2 Negative log-likelihood of generated samples under our PCFG 55

Remerciements

J'aimerais tout d'abord remercier mon directeur de recherche Pr. Yoshua Bengio, pour sa supervision et ses conseils.

Je remercie également mes collaborateurs, sans qui tout ce travail n'aurait pas été possible, Çaglar Gulçehre, Sai Rajeshwar, Sandeeo Subramanian, Adam Trischler, ainsi que Pr. Christopher Pal et Pr. Aaron Courville.

Je remercie également mes collègues et amis du Mila et d'ailleurs, pour toutes les discussions, l'aide et les rires que vous m'avez procurés : Assya Trofimov, Alex Federov, César Laurent, Chin-wei Huang, Devon Hjelm, Dzimitry Badhanau, François Corneau-Tremblay, Frédéric Bastien. Gabriel Huang, Jea Hyun Lim, Joseph Paul Cohen, Martin Weiss, Margaux Luck, Massimo Caccia, Mathieu Germain, Pierre-Luc Carrier, Petar Velickovic, Philippe Lacaille, Samuel Lavoie-Marchildon, Simon Lefrançois, Shawn Tawn, Tristan Sylvain, Yikang Shen, Zhouhan Lin.

Finalement je remercie mon père François ; ma mère Line ; et mon frère Mathieu. Merci pour le support et les encouragements que vous m'avez apportés ces dernières années et tout au long de ma vie.

Introduction

0.1. Apprentissage automatique

0.1.1. Introduction

Un des buts premiers de l'intelligence artificielle est la création d'un modèle qui, déporté dans un environnement complexe, pourra prendre des décisions aussi bonnes que les humains. Les applications possibles sont variées, allant de la segmentation d'images médicales, à la conduite autonome ou la traduction automatique.

Historiquement, des systèmes experts étaient mis en place afin d'y arriver. Toutefois, ces systèmes possédaient d'importants défauts. Ils ne pouvaient pas apprendre sur de nouvelles données, nécessitaient une mise à jour constante par les experts du domaine et généralisaient mal sur des données non conventionnelles. En traduction automatique par exemple, le célèbre rapport d'ALPAC de 1966 note que ces systèmes étaient plus lents, moins précis et deux fois plus chers que la traduction effectuée par des humains [**Hutchins, 2005**].

L'apprentissage automatique est un moyen de contrer ces problèmes. En partant des données brutes, nous pouvons avoir un agent qui *apprend* de lui même les règles nécessaires pour prendre ces décisions. Ce mémoire se concentrera sur l'apprentissage profond (*Deep Learning*), un sous-domaine de l'apprentissage automatique où les modèles sont en mesure d'apprendre des représentations complexes des données afin de prendre de meilleures décisions.

Dans les prochaines sections, nous passerons rapidement sur les bases de l'apprentissage profond et l'apprentissage automatique en général. Nous introduirons également des modèles appliqués aux données séquentielles, un exemple de modèle génératif, ainsi que l'utilisation de décisions discrètes dans ces modèles. De très bonnes références sont disponibles si le lecteur cherche à en savoir plus sur le sujet, notamment [**Goodfellow *et al.*, 2016**]. Dans

le chapitre 1 nous présenterons un article mettant à profit l'utilisation de décisions discrètes dans la prédiction de structures et dans le chapitre 2 nous présenterons un article sur la génération de langues naturelles à l'aide de modèles génératifs.

0.1.2. Type d'apprentissage automatique

Parmi les différents types d'apprentissages, deux types nous intéressent particulièrement dans ce mémoire. **L'apprentissage supervisé** et **l'apprentissage non supervisé**.

En apprentissage supervisé, nous avons un ensemble de données $X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ et des étiquettes qui leur sont associés $Y = \{y^{(1)}, y^{(2)}, ..., y^{(N)}\}$. Chaque $\mathbf{x}^{(i)}$ est composé de caractéristiques (valeur de pixels, taille, etc.) à partir desquelles les décisions sont prises. Notre modèle a pour but de prédire ces étiquettes à partir des observations : $\hat{y}_\theta^{(i)} = f_\theta(\mathbf{x}^{(i)})$, où f_θ est notre modèle paramétrisé par θ.

Le domaine des étiquettes y est également important, puisqu'il définit la fonction de coût utilisée (voir section 0.1.3) et influence grandement le choix du modèle f_θ. En **régression**, nous tentons de prédire un nombre réel, comme la taille d'un individu, ou un indice boursier. Nous avons donc que $y \in \mathbb{R}$. En **classification** nous prédisons dans quelle catégorie appartient un exemple, parmi un choix de C classes. Dans le cas binaire nous avons donc que $y \in \{0, 1\}$, et dans la classification multiple $y \in \mathbf{1}_c$, où $\mathbf{1}_c$ est l'encodage one-hot de l'étiquette. L'encodage one-hot est représenté par un vecteur de taille C, où tous les éléments ont une valeur de 0, sauf à la position c qui a une valeur de 1.

En **apprentissage non supervisé**, seul l'ensemble de données X est disponible. Plusieurs tâches peuvent être accomplies dans cette situation, mais généralement le but est de *comprendre* les données. En clustering par exemple, nous partitionnons les données dans des groupes ou les différents membres $\mathbf{x}^{(i)}$ partagent de caractéristiques. En modélisation et génération de données, nous cherchons plutôt à modéliser leur distribution P_{data}. Plus de détails sur ce sujet dans la section 0.5.

0.1.3. Fonction de coût

La fonction de coût $J(\theta)$ nous permet d'évaluer et d'entraîner efficacement notre modèle f_θ. En apprentissage supervisé, elle prend la forme $J(\theta) = \frac{1}{N}\sum_i^N L(f_\theta(\mathbf{x}^{(i)}), y^{(i)})$, où L détermine à quel point notre prédiction $\hat{y}^{(i)} = f_\theta(\mathbf{x}^{(i)})$ se rapproche de l'étiquette $y^{(i)}$. Idéalement L serait une quantité que nous cherchons à optimiser directement, comme l'erreur de classification, mais comme nous le verrons dans la section 0.2.2, L doit admettre une dérivée non nulle presque partout. Pour obtenir un tel L dans le cas de la classification, imaginons que notre modèle f_θ modélise la distribution conditionnelle $p_\theta(y|\mathbf{x})$. Nous pouvons ainsi reformuler notre problème comme vouloir **maximiser** la probabilité que la classe prédite soit $y^{(i)}$. Formellement si nous supposons que les paires $(\mathbf{x}^{(i)}, y^{(i)})$ sont *i.i.d.*, la vraisemblance des données est :

$$P(Y|X) = \prod_i^N p_\theta(y^{(i)}|x^{(i)}). \tag{0.1.1}$$

Les paramètres θ de notre modèle qui maximisent cette équation sont donc :

$$\theta^* = argmax_\theta P(Y|X) \tag{0.1.2}$$

$$\theta^* = argmax_\theta \prod_i^N p_\theta(y^{(i)}|\mathbf{x}^{(i)}). \tag{0.1.3}$$

Où nous appelons θ^* l'**estimateur de maximum de vraisemblance** (*Maximum Likehood Estimator, MLE*). Comme nous cherchons uniquement θ^*, nous pouvons réécrire l'équation 0.1.3 en prenant le *log* et le remettant à l'échelle :

$$\theta^* = argmax_\theta \big[\sum^N \frac{1}{N} \log p_\theta(y^{(i)}|\mathbf{x}^{(i)})\big]. \tag{0.1.4}$$

Finalement, nous pouvons transformer le problème en une *minimisation* :

$$\theta^* = argmin_\theta \big[\frac{1}{N}\sum^{N} - \log p_\theta(y^{(i)}|\mathbf{x}^{(i)})\big]. \tag{0.1.5}$$

Nous appelons $J(\theta)$ sous cette forme la *negative log-likehood* (NLL). La forme exacte de L dépend donc de la paramétrisation de $p_\theta(y|\mathbf{x})$. En classification binaire, nous supposons que $y^{(i)} \sim Bernouilli(\hat{y}^{(i)})$. L aura alors la forme suivante, appelée l'entropie croisée binaire (*binary cross-entropy*), avec $\hat{y} = p_\theta(y = 1|\mathbf{x})$:

$$\begin{aligned} L(\hat{y}, y) &= -\log p_\theta(y|\mathbf{x}) \\ &= -\log \hat{y}^y(1-\hat{y})^{1-y} \\ &= -\big(y \log \hat{y} + (1-y)\log(1-\hat{y})\big). \end{aligned} \tag{0.1.6}$$

En classification à plusieurs classes nous avons que $y|\mathbf{x},\theta \sim (\hat{y}_1, ..., \hat{y}_C)$, une distribution multinomiale. Dans ce cas-ci nous avons que $L(\hat{y}, y) = -\sum_i^C y_i \log \hat{y}_i$, appelée l'entropie croisée (*cross-entropy*). Dans le cas d'une régression nous avons que $y \sim \mathcal{N}(y; \hat{y}, \sigma^2)$, ce qui nous donne la *Mean square error* : $L(\hat{y}, y) = ||\hat{y} - y||^2$. La paramétrisation exacte pour les différentes fonctions de sortie sera couverte dans la section 0.2.1.

0.2. Réseaux de neurones

Dans cette section, nous définirons une classe spécifique de modèle f_θ, les réseaux de neurones artificiels qui sont la base de l'apprentissage profond.

Les réseaux de neurones tirent leur inspiration de la neuroscience, où chaque neurone est connecté aux neurones de la couche suivante afin de former un réseau. Selon l'information présente dans les neurones d'entrées, l'information est ensuite propagée aux couches suivantes. Des décisions complexes peuvent ensuite être prises selon l'activité des différents neurones. Formellement, nous définissons un neurone comme ceci :

$$a = \sum_i x_i w_i + b$$

$$a = \mathbf{w}^\intercal \mathbf{x} + b$$

Où $\mathbf{x}$ est l'entrée du neurone. Pour chaque caractéristique x_i, nous avons un paramètre w_i qui lui est associé. Nous appelons a la préactivation du neurone. Dans le cas d'un seul neurone nous appelons ceci le perceptron [**Mcculloch et Pitts, 1943**], et la décision est définie par le signe de a.

Toutefois, un seul neurone n'est pas suffisant pour discriminer des exemples non linéairement séparables, même pour des cas simples comme XOR [**Minsky et Papert, 1969**]. Pour résoudre ce type de problèmes, nous devons introduire des **couches cachées**. Nous appelons de tels réseaux des **perceptrons multicouches** (*Multi-layer perceptron, MLP*). Ces couches cachées permettent au réseau d'apprendre des relations non linéaires entre les différentes caractéristiques x_i. Nous appelons cette représentation $\mathbf{h}$ un état *caché*, puisque cette représentation ne fait de sens qu'à l'intérieur du réseau de neurones. À l'aide de cette nouvelle représentation $\mathbf{h}$, le réseau est ensuite en mesure de transformer le problème en un problème linéairement séparable. Aucune restriction n'existe sur la taille de $\mathbf{h}$, ni sur le nombre de couches cachées. En fait, il est démontré qu'avec une seule couche cachée il est possible d'approximer n'importe quelles fonctions lisses [**University of Illinois at Urbana-Champaign *et al.*, 1988**]. Toutefois, en ayant plusieurs couches cachées nous sommes en mesure d'apprendre des représentations de plus en plus abstraites.

Formellement, avec un réseau de neurones f_θ avec une seule couche cachée, nous avons la paramétrisation suivante :

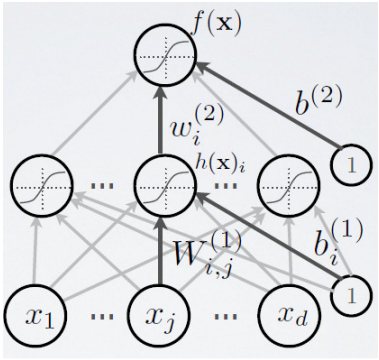


FIGURE 0.1. Représentation graphique des réseaux de neurones avec une couche cachée. Figure tirée de [**Larochelle, 2013**].

$$a_1 = W_1^\intercal \mathbf{x} + \mathbf{b}_1 \tag{0.2.1}$$

$$\mathbf{h}_1 = g_1(\mathbf{a_1}) \tag{0.2.2}$$

$$\mathbf{a}_2 = \mathbf{w}_1^\intercal \mathbf{h}_1 + \mathbf{b}_2 \tag{0.2.3}$$

$$\hat{y} = g_2(\mathbf{a}_2) \tag{0.2.4}$$

Où $\mathbf{a}_1$ et $\mathbf{a}_2$ sont les préactivations des différentes couches. Comme précédemment, une décision est prise selon la valeur de a_2. W_1 est la matrice de paramètres permettant de calculer la représentation cachée $\mathbf{h}_1$. g_1, g_2 sont des fonctions d'activations appliquées à chaque élément. w_2 sont les paramètres permettant de prendre la décision finale. Une représentation graphique est montrée dans la figure 0.1

0.2.1. Fonction d'activation

Une attention particulière doit être apportée au choix de la non-linéarité g dans les réseaux de neurones. Les fonctions d'activations les plus souvent utilisées sont :

$$reLU(x) = max(0, x) \tag{0.2.5}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{0.2.6}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{0.2.7}$$

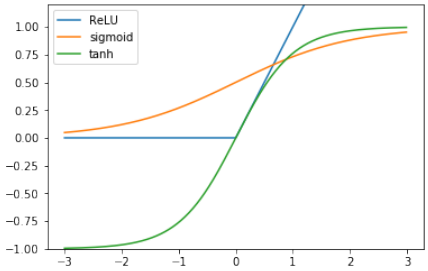


FIGURE 0.2. Représentation graphique des réseaux de neurones multicouche (MLP)

Les différentes fonctions d'activation sont montrées dans la figure 0.2. *ReLU* est généralement plus utilisé puisque le gradient ne sature pas pour la moitié de son domaine, contrairement à *signmoid* et *tanh*. Ceci permet l'utilisation de réseaux plus profonds puisque leurs gradients ont moins tendance à disparaître.

Les fonctions d'activation sont également utilisées à la sortie du réseau afin de paramétriser $p_\theta(y|\mathbf{x})$, comme pour g_2 dans l'équation 0.2.4. Dans le cas de la classification binaire par exemple, $\hat{y} \in \{0,1\}$, la *sigmoid* est utilisé. Dans le cas de la classification multi classes, où $y = \mathbf{1}_c$, le $softmax$ est utilisé :

$$softmax(\mathbf{x})_i = \frac{\exp x_i}{\sum_j^c \exp x_j}. \tag{0.2.8}$$

Cette paramétrisation oblige chacun des éléments à être entre 0 et 1 et force la somme à être égale à 1. Elle peut donc représenter distribution multimodale où $p(y = c|x) = softmax(f_\theta(\mathbf{x}))_c$.

0.2.2. Entraînement

Comme vu dans la section 0.1.3, nous cherchons à minimiser la fonction de coût $J(\theta) = \sum_i^N L(\hat{y}^{(i)}, y^{(i)})$. Dans les réseaux de neurones, ceci est fait en effectuant une descente de gradient, où nous calculons le gradient $\nabla_\theta J$ et suivons sa direction jusqu'à atteindre un minimum. Afin de converger dans un temps raisonnable, nous devons contrôler la taille de ces mises à jour avec un taux d'apprentissage (*learning rate*) α. Un exemple peut être vu dans la figure 0.3.

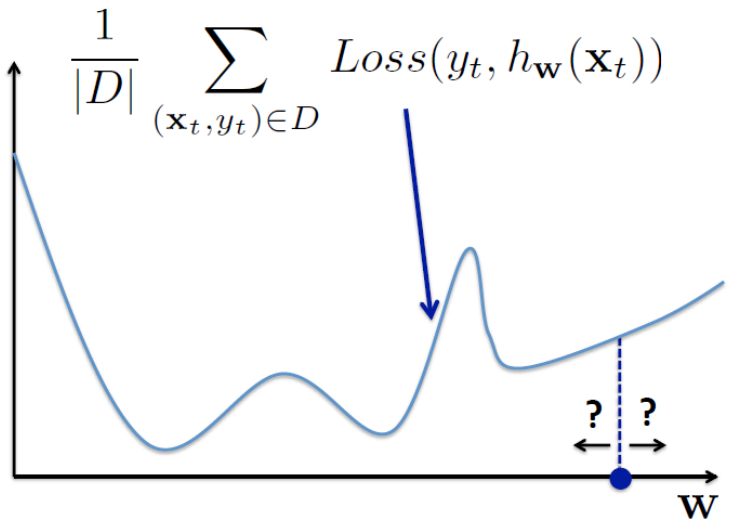


FIGURE 0.3. Descente de gradients dans l'espace des paramètres. Les paramètres sont mis à jour selon la fonction de coût. Figure tirée de [**Larochelle, 2013**].

Pour trouver $\nabla_\theta J$ exactement nous devons calculer $\nabla_\theta L(\hat{y}^{(i)}, y^{(i)})$ pour **tous** les exemples d'entraînement. Toutefois, dans la réalité d'aujourd'hui, le nombre de données disponible est trop grand. ImageNet par exemple, un ensemble d'images sorti en 2010 contient 14 millions [**Deng *et al.*, 2009**]. C'est pourquoi en pratique nous faisons une mise à jour *stochastique* de nos paramètres en estimant ce gradient à partir d'un sous-ensemble aléatoire de données, appeler une *minibatch* : $J(\theta) \approx \sum_i^m L(\hat{y}^{(i)}, y^{(i)})$. La taille m de ces minibatch doit être choisie adéquatement afin de maximiser la parallélisation sans trop ralentir la convergence. L'algorithme résultant, la **descente de gradients stochastique** (*stochastic gradient descent, SGD*), est donc défini comme ceci :

Data: Donnée X, étiquettes Y, taux d'apprentissage α, paramètres initiaux θ_0

Result: Paramètre optimal θ^*

while *Non convergence* **do**

$\Delta_\theta = \nabla_\theta \sum_i^m L(\hat{y}^{(i)}, y^{(i)})$

$\theta = \theta - \alpha\Delta_\theta$

end

Algorithm 1: Descente de gradients stochastique

Plusieurs méthodes existent afin d'accélérer la convergence des modèles. Notamment **momentum** [**Polyak, 1964**] qui accumule les gradients pour chacun des paramètres et accélère

la convergence lorsque la direction d'optimisation moyenne reste la même pour un long moment. **ADAM [Kingma et Ba, 2014]** peut être vu comme une méthode avec momentum, mais qui adapte dynamiquement le taux d'apprentissage pour chacun des paramètres.

0.2.3. Rétropropagation du gradient

Nous avons vu dans la section 0.2.2 que SGD pouvait être utilisée pour trouver la valeur des paramètres qui minimise notre fonction de coût $J(\theta)$. Toutefois, en apprentissage profond la quantité énorme de paramètres et de couches cachées rend difficile le calcul des gradients $\nabla_\theta J$. Pour cette raison, un algorithme a été développé, la **rétropropagation** du gradient (*backpropagation*) **[Rumelhart *et al.*, 1986]** afin d'entraîner efficacement ces modèles. L'algorithme tire profit de la propriété de la dérivation en chaine :

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Où un temps de calcul peut être sauvé en gardant $\frac{dz}{dy}$ en mémoire si plusieurs variables utilisent cette dérivée partielle. Nous pouvons adapter cette propriété au cas vectoriel comme ceci :

$$\frac{\partial z}{\partial x_i} = \sum_i \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x_i} \tag{0.2.9}$$

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^\intercal \nabla_{\mathbf{y}} z \tag{0.2.10}$$

Où $\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^\intercal$ est la Jacobienne de $\mathbf{x}$ par rapport à $\mathbf{y}$. Pour avoir une meilleure intuition de son application dans les réseaux de neurones, nous pouvons regarder les gradients pour les paramètres θ définis en 0.2.4 :

$$\nabla_{w_2} L = (\frac{\partial a_2}{\partial w_2})^\intercal \nabla_{a_2} L \tag{0.2.11}$$

$$\nabla_{b_2} L = (\frac{\partial a_2}{\partial b_2})^\intercal \nabla_{a_2} L \tag{0.2.12}$$

$$\nabla_{a_1} L = (\frac{\partial h_1}{\partial a_1})^\intercal (\frac{\partial a_2}{\partial h_1})^\intercal \nabla_{a_2} L \tag{0.2.13}$$

$$\nabla_{W_1} L = (\frac{\partial a_1}{\partial W_1})^\intercal (\frac{\partial h_1}{\partial a_1})^\intercal (\frac{\partial a_2}{\partial h_1})^\intercal \nabla_{a_2} L \tag{0.2.14}$$

$$\nabla_{b_1} L = (\frac{\partial a_1}{\partial b_1})^\intercal (\frac{\partial h_1}{\partial a_1})^\intercal (\frac{\partial a_2}{\partial h_1})^\intercal \nabla_{a_2} L \tag{0.2.15}$$

Où il est clair que garder $\nabla_{a_2} L$ et $\nabla_{a_1} L$ sauverait énormément de calculs. L'algorithme de rétropropagation consiste donc à calculer itérativement ces gradients en partant de la dernière couche. Concrètement, pour un réseau de neurones avec k couches, nous avons l'algorithme suivant :

Data: paramètres du réseau W, b

Result: $\nabla_W L$, $\nabla_b L$

$\Delta = \nabla_{\hat{y}} L(\hat{y}, y)$

for *k=l, l-1, ... l* **do**

$\Delta = \Delta \odot g'_k(a_k)$; // Calcul de $\nabla_{a_k} L$

$\nabla_{b^k} L = \Delta$; // Calcul de $\nabla_{b_k} L$

$\nabla_{W^k} L = \Delta \; h_{k-1}^\intercal$; // Calcul de $\nabla_{W_k} L$

$\Delta = W_k^\intercal \; \Delta$; // Calcul de $\nabla_{h_{k-1}} L$

end

Algorithm 2: Algorithme de rétropropagation

Nous pouvons voir qu'en gardant en mémoire les différentes représentations cachées h_k, nous sommes en mesure de calculer efficacement les gradients des paramètres pour toutes les couches.

0.2.4. Normalisation dans les réseaux de neurones

Généralement plus les réseaux sont profonds, plus ils sont difficiles à entraîner. Ceci se traduit par la nécessité d'utiliser un taux d'apprentissage plus petit, ce qui ralentit la convergence des réseaux.

Un facteur qui explique ce phénomène est le *covariance-shift interne* [**Ioffe et Szegedy, 2015a**]. Pour bien comprendre le problème, nous devons mettre de l'avant deux propriétés de la rétropropagation : 1. Tous les paramètres sont mis à jour simultanément et 2. Ces mises à jour dépendent de la valeur des autres paramètres dans le réseau. Ceci signifie que pour une couche donnée, ses prochaines entrées ne proviendront plus de la distribution qui avait guidé sa mise à jour. Pour permettre au réseau d'apprendre, nous n'avons d'autre choix que d'utiliser un taux d'apprentissage plus petit afin de prendre en compte ce *shift*.

Un moyen simple pour atténuer le problème est de s'assurer que cette distribution ne change pas au cours de l'entraînement, comme le fait **Batch Normalisation** [**Ioffe et Szegedy, 2015a**]. La technique consiste à normaliser l'entrée de chaque couche cachée afin de contrôler la distribution qui en sort. Idéalement, cette normalisation serait effectuée à travers tous les exemples $x^{(i)}$ de l'ensemble d'entraînement, mais ceci est trop coûteux en pratique. Pour cette raison la normalisation est effectuée à partir de chaque *minibatch*. Formellement, la reparamétrisation prend la forme suivante :

$$\hat{\mathbf{a}}_l^{(i)} = \frac{\boldsymbol{\gamma}_l \odot (\mathbf{a}_l^{(i)} - \boldsymbol{\mu}_l)}{\sigma(\mathbf{a}_l)} + \boldsymbol{\beta}_l$$

$$\boldsymbol{\mu}_l = \frac{1}{m}\sum_i^m \mathbf{a}_l^{(i)}$$

$$\sigma(\boldsymbol{a}_l^{(i)}) = \sqrt{\frac{1}{m-1}\sum_i^m (\mathbf{a}_l^{(i)} - \boldsymbol{\mu}_l)^2}$$

Où $\mathbf{a}_l^{(i)}$ est la préactivation des neurones à la couche l pour l'exemple i. $\boldsymbol{\mu}_l$ et $\sigma(\mathbf{a}_i^l)$ sont la moyenne et l'écart-type empirique des neurones calculés sur la *minibatch* de taille m. Un facteur de gain γ et de biais β est appris pour chaque neurone durant l'entrainement afin de permettre au réseau d'avoir plus de contrôle sur la distribution. Afin de fonctionner durant

la phase d'inférence, où un seul exemple est traité à la fois, nous pouvons garder en mémoire une moyenne mobile de μ_l et σ.

0.3. Données séquentielles

Dans cette section, nous concentrerons sur l'application des réseaux de neurones aux données séquentielles. Formellement, nous avons notre ensemble de données $\mathbf{X} = \{X^{(1)}, X^{(2)}, ...X^{(N)}\}$, où chaque $X^{(i)}$ est à son tour une série de caractéristiques : $X^{(i)} = (\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, ...\mathbf{x}_T^{(i)})$ de longueur variable T. Nous appelons chacun de ces points un pas de temps (*time step*). Dans le cas des langues naturelles, chaque $\mathbf{x}_t$ correspondons à un mot et est représenté par le vecteur one-hot $\mathbf{1}_v$. Dans ce cas ci, $\mathbf{1}_v$ est de la taille du vocabulaire et v est le mot au quel correspond $\mathbf{x}_t$. Nous pouvons avoir le même type de tâche que défini dans la section 0.1.2, comme la classification de documents. Une tâche présente dans les langues naturelles qui tire profit de la nature séquentielle des données est la **modélisation de la langue** (*language modeling*), où le but est de prédire le mot x_t étant donné tous les mots vus précédemment. Notre modèle f_θ se doit donc de modéliser la distribution $p_\theta(\mathbf{x}_t|\mathbf{x}_1, ...\mathbf{x}_{t-1})$. Cette probabilité est paramétrisée par un $softmax$ de la taille du vocabulaire. Prenons par exemple la phrase *En automne les feuilles* ___. Le modèle devra *comprendre* le concept de saison et prédire que le prochain mot à prédire est *tombent*. Pour y arriver, nous entraînons f_θ sur d'énormes quantités de texte (Wikipedia par exemple) dans l'espoir qu'il apprenne de telles relations. L'objectif est encore une fois de minimiser la *negative log-likehood*, en tirant profit de la règle de probabilité en chaîne :

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\log p_\theta(\mathbf{y})) \tag{0.3.1}$$

$$= -\log \prod_i^T p_\theta(\mathbf{y}_i|\mathbf{y}_{<i})) \tag{0.3.2}$$

$$= -\sum_i^T \log p_\theta(\mathbf{y}_i|\mathbf{y}_{<i})) \tag{0.3.3}$$

Où, pour garder une notation consistante, nous assumons que $\mathbf{y}_t = \mathbf{x}_{t+1}$.

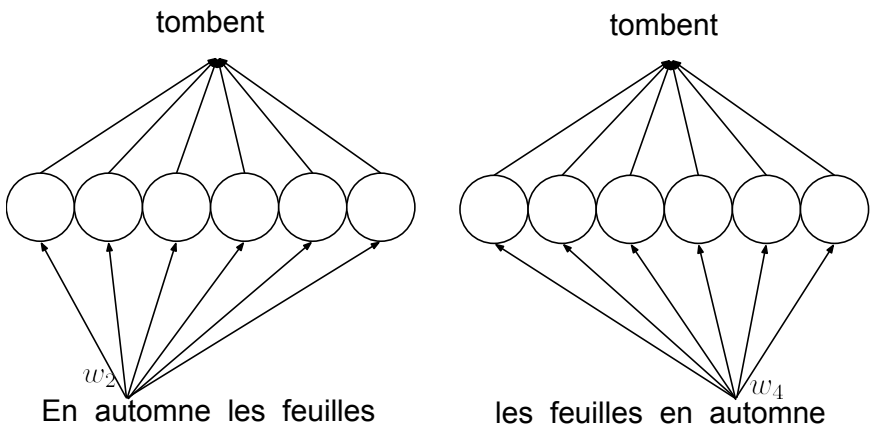


FIGURE 0.4. Un exemple mettant en évidence le problème d'utiliser les perceptrons multicouches sur des données séquentielles. Le modèle doit trouver l'information essentielle, peu importe son emplacement. Une redondance est donc nécessaire entre w_2 et w_4

Pour travailler avec de telles données, les perceptrons multicouches ne conviennent plus. Reprenons notre exemple *En automne les feuilles* ___, et une autre phrase similaire : *les feuilles, en automne,* ___. Afin de prédire le bon mot, f_θ ce doit d'être invariant à la position de l'information pertinente, sinon le modèle devra avoir une énorme redondance dans ces paramètres. L'exemple est présenté dans la figure 0.4 afin d'aider à la visualisation.

Deux principales solutions existent en apprentissage profond afin de pallier ce problème. Les réseaux de neurones récurrents [**Elman, 1990**], [**Jordan, 1986**] et les réseaux de neurones à convolution.

0.3.1. Réseaux de neurones récurrents

Le moyen le plus intuitif pour travailler avec ce type de données est de les traiter séquentiellement. Ainsi, au lieu d'avoir une fonction globale $\mathbf{h} = g(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ qui traite X d'un seul coup, nous introduisons une forme de récurrence. L'information est ainsi accumulée dans l'état caché $\mathbf{h}_t$ au fur et à mesure que la séquence est traitée. Formellement, nous définissons

cette récurrence comme ceci :

$$\mathbf{h}_t = g_t(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t)$$

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

$$\mathbf{h}_t = f(\mathbf{x}_t, f(\mathbf{x}_{t-1}, \mathbf{h}_{t-2}))$$

$$\mathbf{h}_t = f(\mathbf{x}_t, ..., f(\mathbf{x}_1, \mathbf{h}_0))$$

L'avantage comparativement au réseau de neurones traditionnel est que f est réutilisé à chaque étape de la séquence, ce qui permet d'extraire l'information nécessaire, peu importe son positionnement dans la séquence.

Le réseau récurrent "Vanille" $f(\mathbf{x}_t, \mathbf{h}_{t-1})$ a donc la simple forme suivante :

$$\mathbf{a}_t = W_x^\intercal \mathbf{x}_t + W_h^\intercal \mathbf{h}_{t-1} + \mathbf{b} \tag{0.3.4}$$

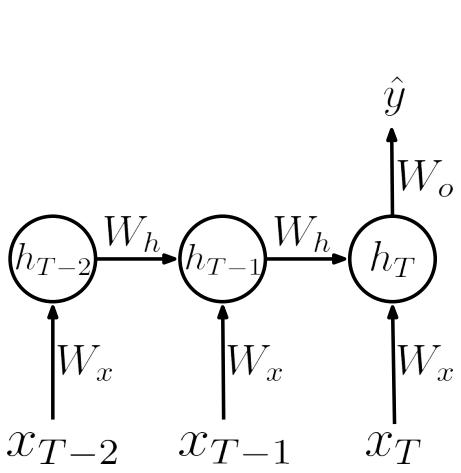
$$\mathbf{h}_t = tanh(\mathbf{a}_t) \tag{0.3.5}$$

$$\tag{0.3.6}$$

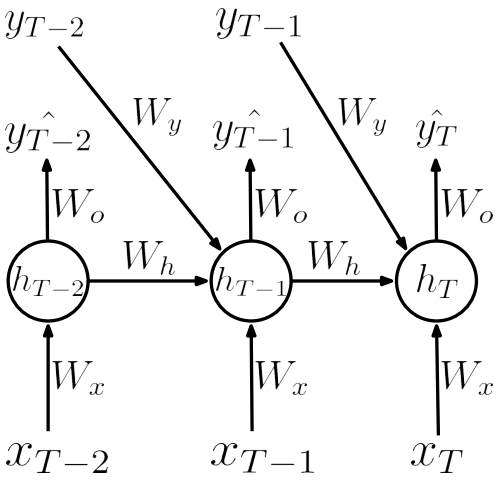
Où $\mathbf{a}_t$ et $\mathbf{h}_t$ sont la préactivation et l'activation au temps t. W_x correspond aux paramètres d'entrées et W_h au paramètre de récurrence. Tous les paramètres sont partagés à travers le temps. h_0 est généralement initialisé à 0. Si nous effectuons une tâche de classification classique, la prédiction $\hat{\mathbf{y}}$ est alors calculée à partir du dernier état $\mathbf{h}_T$:

$$\hat{\mathbf{y}} = W_o^\intercal \mathbf{h}_T + \mathbf{b} \tag{0.3.7}$$

Dans le cas de la modélisation de langues, le réseau reçoit en entrée le dernier élément de la séquence et émet une prédiction à chaque pas de temps :



(a) Réseaux de neurones récurrents, lorsqu'une seule prédiction est effectuée à la fin de la séquence.



(b) Réseaux de neurones récurrents, lorsqu'une prédiction est effectuée à chaque pas de temps, comme en modélisation de la langue.

Figure 0.5. Différentes configurations de réseau de neurones récurrents

$$\mathbf{a}_t = W_x^\intercal \mathbf{x}_t + W_h^\intercal \mathbf{h}_{t-1} + \mathbf{b} \tag{0.3.8}$$

$$\mathbf{h}_t = tanh(\mathbf{a}_t) \tag{0.3.9}$$

$$\hat{y}_t = W_o^\intercal \mathbf{h}_t + \mathbf{b} \tag{0.3.10}$$

Un exemple de chacun des modèles est montré dans la figure 0.5.

0.3.1.1. *Teacher forcing*

Les modèles de langue ou tout autres modèle conditionné sur d'anciennes prédictions sont entraînés avec l'aide du *teacher forcing*. Ceci consiste à donner la vraie valeur $\mathbf{y}_{t-1}$ en entrée au temps t, peu importe la prédiction $\hat{y}_{t-1}$ du modèle. Bien que cela permette d'entraîner efficacement les modèles, un problème survient lors de l'inférence, où les prédictions $\hat{y}_t$ du modèle sont données en entrée. Puisque les modèles ne se sont jamais entraînés en voyant leurs propres prédictions, ils ont tendance à diverger puisque des erreurs de prédiction s'accumulent petit à petit.

0.3.1.2. *Rétropropagation à travers le temps*

L'entraînement des réseaux récurrents se fait également avec l'aide de la descente de gradient et la rétropropagation. Comme nous pouvons le voir dans la figure 0.5, du au partage de paramètres, $\frac{\partial L}{\partial W_x}$ et $\frac{\partial L}{\partial W_h}$ seront une somme à travers le temps. Dans le cas où une seule prédiction est faite, nous avons par exemple que :

$$\nabla_{W_h} L = \sum_{t}^{T} \left(\frac{\partial a_t}{\partial W_h}\right)^{\intercal} \nabla_{a_t} L \tag{0.3.11}$$

La mise à jour se généralise à la modélisation de langues, en sommant à travers les différents pas de temps, comme vu dans l'équation 0.3.3 :

$$\nabla_{W_h} L = \sum_{t}^{T} \nabla_{W_h} L_t \tag{0.3.12}$$

0.3.1.3. *Dépendance à long terme*

Il y toutefois un problème avec les réseaux de neurones récurrents "Vanille" définis plus haut. Ces réseaux ont de la difficulté à apprendre les dépendances à long terme. Ceci se manifeste durant l'entraînement de deux façons : lorsque le gradient explose (causant l'algorithme à diverger) et lorsque le gradient disparaît, ce qui rend impossible d'apprentissage lorsque le signal d'erreur vient de trop loin. Pour avoir une meilleure intuition, prenons une analyse tirée de [**Goodfellow *et al.*, 2016**]. Imaginons que nous avons le réseau de neurones récurrent linéaire suivant :

$$\mathbf{h}_t = W^{\intercal} \mathbf{h}_{t-1} \tag{0.3.13}$$

$$\tag{0.3.14}$$

Où W sont les paramètres de récurrence et $\mathbf{h}$ l'état caché au temps t. Les entrées $\mathbf{x}_t$ et biais $\mathbf{b}$ ont été enlevés pour alléger la notation. En remplaçant récursivement $\mathbf{h}_t$ nous obtenons

$$\mathbf{h}_t = \left(W^t\right)^\intercal \mathbf{h}_0 \tag{0.3.15}$$

En supposant que W puisse être décomposé en vecteurs propres Q et valeurs propres Λ, nous avons que :

$$\mathbf{h}_t = \left(Q^\intercal \Lambda Q\right)^t \mathbf{h}_0 \tag{0.3.16}$$

$$\mathbf{h}_t = Q^\intercal \Lambda^t Q \mathbf{h}_0 \tag{0.3.17}$$

Où Q est orthogonal, et Λ une matrice diagonale. Puisque les valeurs propres Λ sont élevées à la puissance t, les vecteurs propres < 1 disparaissent exponentiellement vite. Comme nous l'avons vu dans la section 0.3.1.2, $\nabla_W L$ est une somme sur tout les pas de temps. Il est donc clair que les dépendances à long terme (c.-à-d. loin de T) auront un impact négligeable lors de la mise à jour des paramètres. Dans le cas où les valeurs propres sont > 1, les gradients explosent, ce qui risque de faire diverger la procédure d'optimisation.

Ce problème a été longuement étudié, notamment dans [**Bengio *et al.*, 1994**] et [**Hochreiter, 1991**], ainsi que revu dans [**Pascanu *et al.*, 2013b**].

Dans le cas de gradients explosifs, la solution proposée par [**Pascanu *et al.*, 2013b**] est de limiter la norme maximale du gradient lors de l'entraînement. Bien qu'une valeur maximale doit être définie à l'avance, les auteurs montrent qu'en pratique cette méthode est très résiliente. Lorsque les gradients disparaissent, les choses se compliquent. [**Bengio *et al.*, 1994**] démontre que pour être en mesure d'apprendre adéquatement des relations à long terme, les paramètres devront entrer dans une zone où les gradients disparaîtront.

0.3.1.4. *Réseau récurrent à mémoire court et long terme*

Un moyen de permettre au réseau d'apprendre des dépendances à long terme est l'utilisation de portes (*gates*), qui facilite le flux d'information. Une modification mineure au réseau "Vanille" définie en 0.3.6 nous donnerait :

$$\begin{aligned} \mathbf{a}_t &= W_x^\intercal \mathbf{x}_t + W_h^\intercal \mathbf{h}_{t-1} + \mathbf{b} \\ \tilde{\mathbf{h}}_t &= tanh(\mathbf{a}_t) \\ \mathbf{h}_t &= \boldsymbol{\alpha}_\theta \odot \tilde{\mathbf{h}}_t + (1 - \boldsymbol{\alpha}_\theta) \odot \mathbf{h}_{t-1} \end{aligned} \tag{0.3.18}$$

Où $\alpha_i \in [0, 1]$ est la porte qui peut être apprise par le modèle et dicte la quantité d'information à ajouter à l'état caché h_i. Dans cette situation $\mathbf{h}_t$ est une combinaison convexe entre l'état précédent $\mathbf{h}_{t-1}$ et l'état candidat $\tilde{\mathbf{h}}_t$. Si les bonnes valeurs pour $\boldsymbol{\alpha}$ sont choisies, le modèle sera en mesure de mémoriser de l'information observée sur une longue période. De plus, comme $\mathbf{h}_t$ dépend linéairement de $\mathbf{h}_0$, le gradient a moins tendance à disparaître. Nous appelons ceci des *leaky units* [**Mozer, 1992**].

Un modèle extrêmement populaire qui met à profit ce type de porte est le réseau récurrent à mémoire court et long terme (*Long Short term memory, LSTM*) [**Hochreiter et Schmidhuber, 1997**]. La fonction de récurrence f_θ est définie comme ceci :

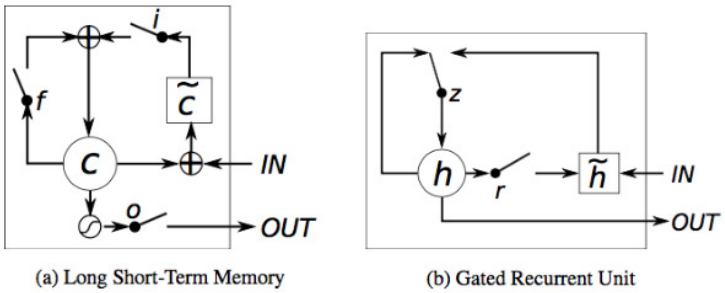


Figure 0.6. Comparaison entre la cellule de récurrence du LSTM et du GRU. Les différentes portes sont mises en évidence. Figure tirée de [**Chung *et al.*, 2014**].

$$\mathbf{i}_t = \sigma(U_i^\intercal \mathbf{x}_t + W_i^\intercal \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{0.3.19}$$

$$\mathbf{f}_t = \sigma(U_f^\intercal \mathbf{x}_t + W_f^\intercal \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{0.3.20}$$

$$\mathbf{o}_t = \sigma(U_o^\intercal \mathbf{x}_t + W_o^\intercal \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{0.3.21}$$

$$\tilde{\mathbf{c}}_t = \tanh(U_h^\intercal \mathbf{x}_t + W_h^\intercal \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{0.3.22}$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \tilde{\mathbf{c}}_t \odot \mathbf{i}_t \tag{0.3.23}$$

$$\mathbf{h}_t = tanh(\mathbf{c}_t) \odot \mathbf{o}_t \tag{0.3.24}$$

Où i, f et o sont les portes qui bloquent ou autorisent le flux d'information à travers le temps. $\mathbf{i}$ dicte combien d'information sera ajoutée (*input gate*), $\mathbf{f}$ dicte l'information qui sera retenue (*forget gate*), et $\mathbf{o}$ dicte l'information qui sera transmise au prochain pas de temps (*output gate*). $\mathbf{c}_t$ est une mémoire que le réseau peut modifier durant le traitement des données. La partie essentielle est l'équation 0.3.23, où nous remarquons que la cellule $\mathbf{c}_t$ ne passe à travers aucune non-linéarité tout au long du traitement de la séquence. $\mathbf{f}_t$ et $\mathbf{i}_t$ prennent donc le rôle de la porte $\boldsymbol{\alpha}$ définie en 0.3.18.

0.3.1.5. *Gated Recurent Units*

Une méthode alternative aux LSTM est le *Gated Recurrent Unit* (GRU) [**Cho *et al.*, 2014b**]. La fonction d'activation de ce dernier peut être exprimée comme suit :

$$\mathbf{z}_t = \sigma(W_z^\intercal \mathbf{x}_t + U_z^\intercal \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{0.3.25}$$

$$\mathbf{r}_t = \sigma(W_r^\intercal \mathbf{x}_t + U_r^\intercal \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{0.3.26}$$

$$\tilde{\mathbf{h}}_t = \sigma(W_h^\intercal \mathbf{x}_t + U_h^\intercal (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{0.3.27}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{0.3.28}$$

Où $\mathbf{z}_t$ et $\mathbf{r}_t$ sont les portes qui dictent le flux d'information. Comme pour le LSTM, le fait que $\mathbf{h}_t$ dépendent linéairement de $\mathbf{h}_0$ est ce qui permet au réseau d'apprendre des dépendances à long terme. La différence principale entre les deux modèles est que le réseau n'a pas l'opportunité de cacher une partie de son état aux futurs pas de temps. Une représentation graphique peut être vue dans la figure 0.6.

Bien le GRU soit plus rapide à exécuter, aucune différence notable n'a été trouvée entre les deux modèles [**Chung *et al.*, 2014**].

0.3.1.6. *Normalisation dans les réseaux récurrents*

Comme décrit dans la section 0.2.4, le *covariance shift interne* impacte beaucoup l'optimisation des réseaux profonds. Ceci n'exclut pas les réseaux récurrents. Toutefois, contrairement au réseau de neurones traditionnel, la taille variable des données d'entrées cause un problème. Des techniques ont été mises au point afin d'adapter *batch normalization* [**Ioffe et Szegedy, 2015a**] au RNN, dont [**Laurent *et al.*, 2016**], [**Cooijmans *et al.*, 2016**]. [**Cooijmans *et al.*, 2016**] en particulier montre qu'avec une bonne paramétrisation, nous sommes en mesure d'appliquer *batch normalisation* au réseau de neurones récurrent. Une autre technique, utilisée dans le chapitre 1, est *layer normalization* [**Ba *et al.*, 2016**]. Concrètement, au lieu de normaliser chaque préactivation $\mathbf{a}$ selon les différents exemples ou pas de temps, *layer normalization* normalise sur l'axe des neurones eux-mêmes. Chaque exemple est donc normalisé individuellement et nous n'avons pas à nous inquiéter de la différence de longueur entre les différents exemples de la *minibatch*. Formellement, nous avons que :

$$\hat{a}_i^l = \frac{\gamma_i^l(a_i^l - \mu^l)}{\sigma(a^l)} + \beta_i$$
$$\mu^l = \frac{1}{H}\sum_i a_i^l$$
$$\sigma(a^l) = \sqrt{\sum_i (a_i^l - \mu^l)^2}$$

Où H est la taille de la couche cachée. Comme pour *batch normalisation*, le réseau est en mesure de contrôler la distribution à la sortie de chaque couche avec les paramètres de gains γ_i et β_i.

Les auteurs ont montré que *layer normalisation* permet d'avoir de meilleurs résultats et de converger plus vite sur une multitude de tâches en traitement des langues naturelles. Dans le cas du LSTM et du GRU, nous pouvons normaliser la préactivation de $\mathbf{h}_t$ et $\mathbf{c}_t$.

0.3.2. Réseaux de neurones à convolution

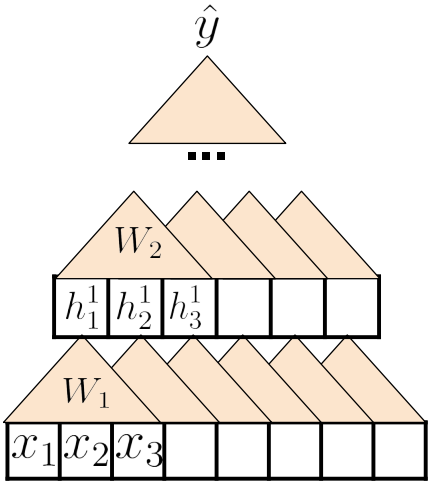
Les réseaux de neurones à convolution (*Convolution neural network, CNN*) peuvent aussi être utilisés pour traiter les données séquentielles. [**Waibel *et al.*, 1990**] notamment, applique des CNN à la classification de phonèmes. Plus récemment *Wavenet* effectue de la génération vocale [**Van Den Oord *et al.*, 2016**].

Supposons que nous avons notre entrée $\mathbf{x}$, ainsi que notre noyau de convolution $\mathbf{w}$. La convolution en 1D ($\mathbf{x} * \mathbf{w}$) est définie comme suit :

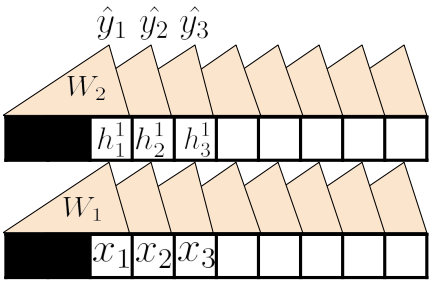
$$a_t = \sum_{i=-\inf}^{\inf} x_i w_{t-i}$$

Où a_t est le résultat après la convolution à l'emplacement t, et correspond à la préactivation dans le cas des réseaux de neurones. Puisque la convolution est commutative et qu'en pratique $\mathbf{w} \in \mathbb{R}^H$ est de taille finie, nous pouvons obtenir la formule plus intuitive suivante :

$$a_t = \sum_{i=0}^{H} x_{t-i} w_i$$



(a) Réseau de neurones à convolution pour une tâche de classification. Chaque h_i^l n'est calculé qu'à partir d'information locale. Aucun sous-échantillonnage n'est utilisé pour cet exemple.



(b) CNN appliqué à la tâche de modélisation de langue. Chaque $\hat{y}_t$ n'est prédit qu'à partir des éléments précédents $x_{<t}$.

Figure 0.7. Différente configuration de réseaux de neurones à convolutions

Où H est la taille du filtre. Traditionnellement plusieurs noyaux $\mathbf{w}$ sont utilisés simultanément, afin d'extraire différentes caractéristiques. Nous voyons clairement que a_i (appelé *feature map* en Anglais) n'est calculé qu'à partir d'information locale, dont la taille est déterminée par H. Lors du traitement des langues naturelles, H correspond à la taille des N-grams utilisés pour extraire l'information. Afin d'apprendre des dépendances à long terme tout en gardant le partage de paramètres au maximum, nous pouvons empiler des couches cachées, ou encore faire du sous-échantillonnage en effectuant du *pooling*, ou du *striding*. Un exemple est montré dans la figure 7(a).

Dans le cas des modèles de langues comme dans le chapitre 2, nous devons nous assurer que le modèle ne puisse pas "voir dans le futur", comme nous le voyons en 0.3.3. Pour y parvenir, nous pouvons masquer les filtres $\mathbf{w}$ **[Oord *et al.*, 2016]**, ce qui se traduit en pratique en ajoutant des zéros de taille $H-1$ au début de la séquence. Encore une fois, nous ajoutons des couches afin d'apprendre des dépendances à long terme. Un exemple est montré dans la figure 7(b). Les modèles sont entrainés avec le *teacher forcing* couvert en 0.3.1.1. Un avantage que les CNNs ont sur les données séquentielles par rapport aux RNNs,

est la possibilité de paralléliser le traitement de $\mathbf{x}$. En effet, contrairement au RNNs, le calcul de $\mathbf{h}_t$ ne dépend pas de $\mathbf{h}_{t-1}$.

0.4. Modèles séquence-à-séquence

Une autre classe de problème important est les problèmes de séquence-à-séquence (*Sequence-to-sequence*). Formellement, nous avons notre entrée $X^{(i)} = (\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, ...\mathbf{x}_{T_x}^{(i)})$, et l'étiquette qui lui est associée est également une séquence $Y^{(i)} = (\mathbf{y}_1^{(i)}, \mathbf{y}_2^{(i)}, ...\mathbf{y}_{T_y}^{(i)})$ d'une longueur arbitraire. Plusieurs tâches correspondent à cette description, mais deux nous intéressent tout particulièrement dans ce mémoire : la traduction automatique et la compréhension des langues, discutées plus en détail dans la section 0.4.2 et le chapitre 1.

Afin d'utiliser les réseaux de neurones sur ce type de tâches, [**Cho *et al.*, 2014c**] , [**Sutskever *et al.*, 2014a**] ont introduit le concept de modèle séquence-à-séquence, ou encodeur-décodeur, utilisant deux RNNs. L'**encodeur** résume la séquence d'entrée X dans un vecteur **contexte** $\boldsymbol{\psi}$ de taille fixe, et un **décodeur** qui génère la séquence cible $\hat{\mathbf{y}}$, conditionné sur $\boldsymbol{\psi}$. Similaire à la tâche de modélisation de langue, la sortie du décodeur $\hat{\mathbf{y}}_t = f_{decodeur}(\mathbf{h}_{t-1}^{dec}, \hat{\mathbf{y}}_{t-1}, \boldsymbol{\psi})$ est donnée par :

$$\mathbf{a}_t = W_x^\intercal \hat{\mathbf{y}}_{t-1} + W_h^\intercal \mathbf{h}_{t-1}^{dec} + W_c^\intercal \boldsymbol{\psi} + \mathbf{b} \tag{0.4.1}$$

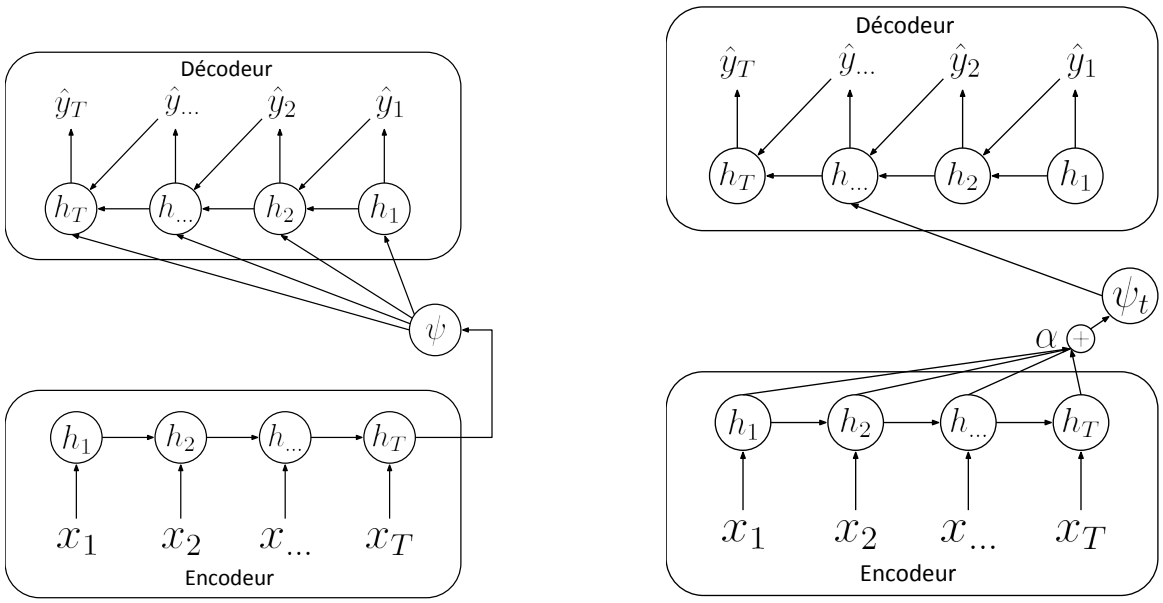
$$\mathbf{h}_t^{dec} = tanh(\mathbf{a}_t) \tag{0.4.2}$$

$$\hat{\mathbf{y}}_t = W_o^\intercal \mathbf{h}_t^{dec} + \mathbf{b} \tag{0.4.3}$$

Où $\mathbf{h}_t^{dec}$ est l'état caché du décodeur au temps t et le vecteur contexte $\boldsymbol{\psi} = \mathbf{h}_T^{enc}$ est le dernier état caché de l'encodeur. Un exemple est montré dans la figure 8(a). Nous pouvons également généraliser ceci à l'utilisation de CNN, comme présenté dans [**Gehring *et al.*, 2017**].

0.4.1. Séquence-à-séquence avec attention

Un problème avec le modèle de la section précédente, est que l'encodeur se doit de résumer toute l'information dans un vecteur de taille fixe et statique. Ceci peut être suffisant lorsque



(a) Modèle Encodeur-Décodeur. Le décodeur génère la séquence cible $\hat{y}$ conditionnée par le vecteur contexte c (fixe) produit par l'encodeur.

(b) Modèle Encodeur-Décodeur avec attention. Chaque vecteur contexte c_t est généré dynamiquement selon l'état du décodeur.

Figure 0.8. Comparaison des modèles Encodeur-Décodeur avec et sans attention.

les séquences sont relativement courtes, mais plus la séquence X est longue, plus le réseau se doit de compresser de l'information. La solution proposée par [**Bahdanau *et al.*, 2014a**] est d'utiliser un mécanisme d'**attention** dans le décodeur. Intuitivement, avant chaque prédiction $\hat{\mathbf{y}}_t$, le décodeur choisit dynamiquement où porter son attention dans la séquence d'entrée afin de calculer le contexte $\boldsymbol{\psi}_t$. Formellement, nous avons la même équation que définie en 0.4.3, mais où $\boldsymbol{\psi}_t$ est calculé comme suit :

$$\boldsymbol{\psi}_t = \sum_{i}^{T_{\mathbf{x}}} \alpha_i h_i^{enc} \tag{0.4.4}$$

$$\boldsymbol{\alpha} = softmax(\hat{\boldsymbol{\alpha}}) \tag{0.4.5}$$

$$\hat{\alpha}_i = f(h_i^{enc}, h_{t-1}^{dec}) \tag{0.4.6}$$

Où f est le mécanisme d'attention, un perceptron multicouche dans le papier original [**Bahdanau *et al.*, 2014a**]. Ce mécanisme compare chaque sortie $\mathbf{h}_t^{enc}$ de l'encodeur avec

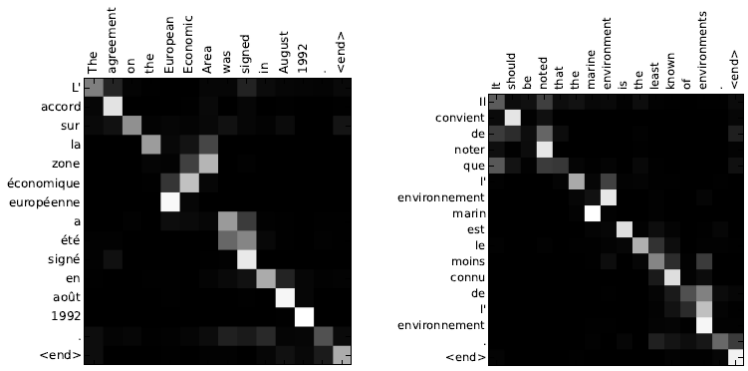


Figure 0.9. Alignement des mots entre les phrases sources et cible. Figure tirée de [**Bahdanau *et al.*, 2014a**].

l'état caché du décodeur. $\boldsymbol{\alpha}$ est le vecteur d'attention sommant à 1 et représente où le décodeur regarde à chaque pas de temps. Une comparaison graphique des deux modèles peut être vue dans la figure 0.8.

Un effet intéressant du mécanisme d'attention est qu'il nous procure un moyen simple d'interpréter les décisions du modèle. Les auteurs de [**Bahdanau *et al.*, 2014a**] remarquent que le réseau apprend à aligner les mots d'entrée et de sortie, comme nous pouvons le voir dans la figure 0.9. Plusieurs articles tentent d'améliorer cette attention/alignement, y compris celui présenté dans le chapitre 1.

0.4.2. Application

Dans cette section nous décrirons brièvement les principales applications des modèles séquence-à-séquence décrites plus haut : la traduction automatique et la compréhension de langues.

En **traduction automatique**, le but est de traduire une phrase d'un domaine *source* à un domaine *cible*. Par exemple du Français à l'Anglais, ou de L'Anglais à l'Allemand. Généralement cette traduction se fait mot à mot, mais la traduction par sous-mots (ou encore par caractères) est de plus en plus présente. Ceci évite d'avoir un vocabulaire trop gros et permet de générer les mots hors de notre vocabulaire, comme nous le verrons dans le chapitre 1. Pour évaluer les traductions d'un modèle, le **Bleu Score** est utilisé [**Papineni *et al.*, 2002**]. Cette métrique calcule la précision obtenue par le modèle lorsque sa prédiction $\hat{\mathbf{y}}$ est comparée à des références humaines $\mathbf{y}$. Cette précision est calculée à partir des différents N-gram

(généralement de 1-gram à 4-gram) présent dans $\hat{\mathbf{y}}$ et $\mathbf{y}$. La métrique s'assure également de pénaliser les traductions trop courtes et les traductions dégénérées.

La **Compréhension de textes**, et tout particulièrement le *question answering* est une autre tâche où les modèles d'Encodeur-Décodeur peuvent être utilisés. Imaginons par exemple le contexte suivant : *Augustus fut le premier empereur de Rome, suivi par Tiberius.* Ainsi que la question : *Quel est le nom du deuxième empereur de Rome ?*. Afin de répondre adéquatement à la question, le modèle se doit de **raisonner**. Un ensemble de données populaire pour cette tâche est SQUAD [**Rajpurkar *et al.*, 2016a**], où étant donné un contexte et une question comme l'exemple précédent, le modèle doit trouver quel passage répond à la question dans le contexte initial. Une autre tâche de compréhension de textes est la génération de questions, que nous explorons plus en détail dans le chapitre 1.

0.5. Modèles génératifs

Jusqu'à présent, nous nous sommes principalement concentrés sur l'apprentissage supervisé, où la tâche est de prédire le $y^{(i)}$ associé à chaque $\mathbf{x}^{(i)}$. Comme expliqué dans la section 0.1.2, lorsque ces étiquettes ne se sont pas présentes, nous faisons de l'**apprentissage non supervisé**, où nous pouvons entre autres entraîner des modèles génératifs. Le but de ces modèles est de modéliser la distribution $p_{data}(\mathbf{x})$, où pour y arriver, nous assumons que les données $\mathbf{x}$ sont générées à partir de variables *latentes* $\mathbf{z}$: $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$. Modéliser $P(\mathbf{x}|\mathbf{z})$ correctement signifierait que le modèle comprend bien "le monde" qui a généré ces données. La prochaine section présentera les Réseaux Antagonistes Génératifs (*Generative Adversarial network, GAN*) [**Goodfellow *et al.*, 2014**], utilisés dans le chapitre 2. Pour plus détails sur les autres modèles génératifs, notamment les *Boltzmann Machines* et les autoencodeurs variationnels, nous référons [**Goodfellow *et al.*, 2016**].

0.5.1. Réseaux Antagonistes Génératifs

Dans le cas des GAN [**Goodfellow *et al.*, 2014**], le but est de directement générer des échantillons provenant de $P(\mathbf{x})$ à partir de la variable latente $\mathbf{z} \sim P(\mathbf{z})$. La distribution $p_{data}(\mathbf{x})$ n'est donc jamais modélisée directement. Les GAN sont constitués de deux réseaux

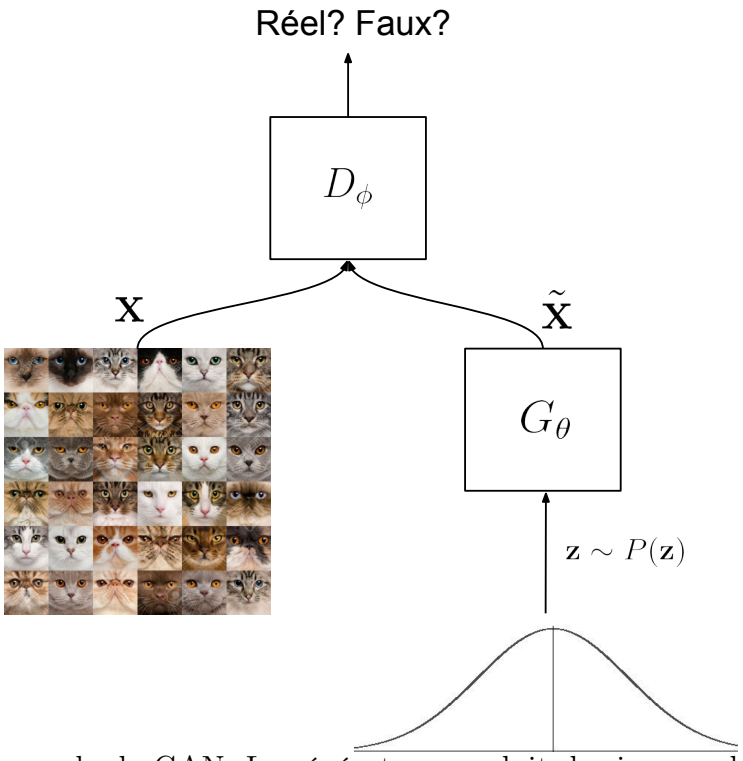


FIGURE 0.10. Exemple de GAN. Le générateur produit des images de chats et le discriminateur détermine si elles sont réalistes ou non.

de neurones. Un **Générateur** $G_\theta(\mathbf{z})$ qui génère les échantillons $\tilde{\mathbf{x}} \sim G_\theta(\mathbf{z})$, ainsi qu'un **Discriminateur** $D_\phi(\mathbf{x})$. Le rôle de D_ϕ est de discriminer les $\mathbf{x}$ selon s'ils sont réels ($\mathbf{x} \sim P_{data}$), ou faux ($\tilde{\mathbf{x}} \sim G_\theta(\mathbf{z})$). Le rôle du générateur est donc de tromper ce discriminateur en générant des $\tilde{\mathbf{x}}$ le plus réaliste possible. Le moyen classique de formuler l'interaction entre les deux modèles est à travers un jeu à somme nulle, où chaque joueur maximise ou minimise la récompense $v(G_\theta, D_\phi)$. Formellement, nous avons que :

$$\min_{G_\theta} \max_{D_\phi} v(G_\theta, D_\phi) = \min_{G_\theta} \max_{D_\phi} \mathbb{E}_{x \sim p_{data}}[\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(z)}[\log(1 - D_\phi(G_\theta(\mathbf{z}))] \quad (0.5.1)$$

Où généralement la variable latente $\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{1})$. L'entraînement du GAN alterne donc entre la mise à jour du discriminateur et du générateur. Un exemple est montré dans la figure 0.10.

Les auteurs montrent également qu'à l'équilibre de Nash, avec un D_ϕ et G_θ optimal, les échantillons $\tilde{\mathbf{x}} \sim G_\theta(\mathbf{z})$ proviennent de la distribution P_{data}. Toutefois, un désavantage des

GAN est leur incapacité à calculer la densité $P(\mathbf{x})$, ce qui rend l'évaluation et la comparaison de ces modèles extrêmement difficile. Plusieurs problèmes peuvent également survenir lors de l'entraînement de ces modèles. Le générateur peut par exemple se dégénérer et ne produire qu'une très petite variété d'échantillons $\tilde{\mathbf{x}}$, un problème que nous appelons *mode collapse*. Les différentes techniques développées pour pallier ce problème seront décrites plus en détail dans le chapitre 2.

0.6. Décision discrète

Supposons maintenant que les représentations cachées h_i puissent être stochastiques et discrètes. Un état discret permet de travailler dans des domaines comme les langues naturelles, comme nous le verrons dans le chapitre 2. Ceci rend également possible le calcul conditionnel, comme mis de l'avant dans [**Bengio *et al.*, 2013a**], [**Bengio, 2013**], ainsi que dans le chapitre 1.

Un exemple simple d'état discret serait le cas binaire $h_i \sim Bernouilli(p_\theta)$ où p_θ serait appris :

$$a_i = \mathbf{w}_i^\intercal \mathbf{x} \tag{0.6.1}$$

$$p_\theta = sigmoid(a_i) \tag{0.6.2}$$

$$h_i = \begin{cases} 1 & \text{if } z_i > p_\theta \\ 0 & \text{sinon} \end{cases} \tag{0.6.3}$$

Où $z_i \sim U[0,1]$ et la fonction en escalier est utilisée comme fonction d'activation. Le problème est que tous les réseaux discutés jusqu'à maintenant utilisaient la descente de gradient pour la mise à jour des paramètres. Ceci interdit donc l'utilisation des fonctions discrétisante comme dans l'exemple ci-haut, puisque $\frac{dh_i}{\partial a_i}$, et par conséquent $\frac{\partial J(h_i)}{\partial a_i}$, égal 0 presque en tous points.

0.6.1. REINFORCE

L'algorithme de REINFORCE [**Williams, 1992a**] part de l'observation que bien que $\frac{\partial J(h_i)}{\partial a_i} = 0$ pour un seul échantillon, la *moyenne* $\frac{\partial \mathbb{E}_{h_i}[J(h_i)]}{\partial a_i}$ est non nulle. Formellement, les auteurs montrent que :

$$\frac{\partial \mathbb{E}_{h_i}[J(h_i)]}{\partial a_i} = \frac{\partial}{\partial a_i} \sum_{h_i} p(h_i) J(h_i) \tag{0.6.4}$$

$$= \sum_{h_i} J(h_i) \frac{\partial p(h_i)}{\partial a_i} \tag{0.6.5}$$

$$= \sum_{h_i} J(h_i) p(h_i) \frac{\partial \log p(h_i)}{\partial a_i} \tag{0.6.6}$$

$$= \mathbb{E}_{h_i}\big[J(h_i) \frac{\partial \log p_\theta}{\partial a_i}\big] \tag{0.6.7}$$

Où l'équation 0.6.6 est obtenue en remarquant que $\frac{\partial \log f(x)}{\partial z} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial z}$. Puisque $\frac{\partial \log p_\theta}{\partial a_i} \neq 0$, nous sommes donc en mesure d'estimer ce gradient. Toutefois, la variance de cette estimation peut être très grande, ce qui ralentit la convergence des modèles. Pour réduire cette variance, nous pouvons introduire un **cadre de référence** (*baseline*) r :

$$\frac{\partial \mathbb{E}_{h_i}[J(h_i)]}{\partial a_i} = \mathbb{E}_{h_i}\big[(J(h_i) - r_t) \frac{\partial \log p_\theta}{\partial a_i}\big] \tag{0.6.8}$$

Dans le cas le plus simple, r est une moyenne mobile qui traque le changement de la perte $J(\theta)$ au cours de l'entraînement :

$$r_t = \gamma J(\theta)_{t-1} + (1 - \gamma) r_{t-1} \tag{0.6.9}$$

Où $0 < \gamma < 1$. Dans le cas binaire présenté en 0.6.3 où p_θ est paramétrisé par une *sigmoid*, la mise à jour de w_{ij} est donnée par :

$$w_{ij} = w_{ij} - \alpha \frac{\partial \mathbb{E}_{h_i}[J(h_i)]}{\partial a_i} \frac{\partial a_i}{w_{ij}} \tag{0.6.10}$$

$$= w_{ij} - \alpha \mathbb{E}_{h_i}\big[(J(h_i) - r_t) \frac{\partial log p_\theta}{\partial a_i}\big] \frac{\partial a_i}{\partial w_{ij}} \tag{0.6.11}$$

$$= w_{ij} - \alpha (J(h_i) - r_t)(h_i - p_\theta) x_i \tag{0.6.12}$$

0.6.2. Straight-Through Estimator

Une solution plus simple encore est le *Straight-Throught Estimator* [**Bengio *et al.*, 2013a**]. La méthode consiste à ignorer l'étape de discrétisation lors du calcul de la rétropropagation, en supposant que $h_i = a_i$ et donc que $\frac{\partial h_i}{\partial a_i} = 1$. Bien que cette technique soit biaisée, les auteurs ont montré qu'elle marchait bien en pratique.

0.6.3. Gumbel-softmax

Dans le cas où l'état h_i provient d'une distribution multinomiale $[p_1, ...p_C]$ avec C catégories, nous pouvons utiliser le *Gumbel-Sofmax reparametrisation trick* [**Maddison *et al.*, 2016**] [**Jang *et al.*, 2016**]. Similairement à 0.6.3, nous avons que h_i est paramétrisé comme ceci :

$$\mathbf{a} = \mathbf{W}^\intercal \mathbf{x} \tag{0.6.13}$$

$$\mathbf{p}_\theta = softmax(\mathbf{a} + \mathbf{z})/\tau) \tag{0.6.14}$$

$$\mathbf{h}_i = \mathbf{p}_\theta \tag{0.6.15}$$

Où $z_i \sim Gumbel(0,1)$ et chaque élément a_i paramétrise le logit du p_i correspondant. τ représente la température et contrôle la souplesse de la distribution : si τ s'approche 0, $\mathbf{p}$ se rapproche d'un vecteur one-hot, et plus τ s'en éloigne, plus $\mathbf{p}$ devient uniforme. $\mathbf{p}$ est ensuite passée directement en entrée au reste du réseau. Puisque $\mathbf{p}$ est continue, nous pouvons utiliser l'algorithme de rétropropagation pour l'entraînement du modèle.

Chapitre 1

Planification dans les modèles séquence-à-séquence

Plan, Attend, Generate : Planning for Sequence-to-Sequence Models

Francis Dutil*, Caglar Gulcehre*, Adam Trischler, Yoshua Bengio [1]

Contribution personelle L'idée d'incorporer de la planification dans le module d'attention nous est venue à Caglar et moi lors d'une séance de remue-méninges. Caglar à mis au point l'architecture originale pour PAG et $rPAG$ et je me suis chargé de son implémentation pour les tâches de traduction automatique et d'algorithmie, aidé par Caglar pour l'optimisation et débogage du code. J'ai également roulé la majorité des expériences sur les tâches de traduction automatique et de génération de question, aidé par Caglar sur le choix des hyper paramètres et diverses astuces d'optimisation. Pour l'écriture de l'article, j'ai principalement aidé pour la section des expériences.

Affiliations

Francis Dutil*, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Caglar Gulcehre*, MILA, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Adam Trischler, Microsoft Research

Yoshua Bengio, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CIFAR Senior Fellow

1. * Signifie une contribution égale au projet

1.1. Abstract

We investigate the integration of a planning mechanism into sequence-to-sequence models using attention. We develop a model which can plan ahead in the future when it computes its alignments between input and output sequences, constructing a matrix of proposed future alignments and a commitment vector that governs whether to follow or recompute the plan. This mechanism is inspired by the recently proposed strategic attentive reader and writer (STRAW) model for Reinforcement Learning. Our proposed model is end-to-end trainable using primarily differentiable operations. We show that it outperforms a strong baseline on character-level translation tasks from WMT'15, the algorithmic task of finding Eulerian circuits of graphs, and question generation from the text. Our analysis demonstrates that the model computes qualitatively intuitive alignments, converges faster than the baselines, and achieves superior performance with fewer parameters.

1.2. Introduction

Several important tasks in the machine learning literature can be cast as sequence-to-sequence problems[**Cho *et al.*, 2014b, Sutskever *et al.*, 2014b**]. Machine translation is a prime example of this : a system takes as input a sequence of words or characters in some source language, then generates an output sequence of words or characters in the target language – the translation.

Neural encoder-decoder models[**Cho *et al.*, 2014b**], [**Sutskever *et al.*, 2014b**] have become a standard approach for sequence-to-sequence tasks such as machine translation and speech recognition. Such models generally *encode* the input sequence as a set of vector representations using a recurrent neural network (RNN). A second RNN then *decodes* the output sequence step-by-step, conditioned on the encodings. An important augmentation to this architecture, first described by[**Bahdanau *et al.*, 2015**], is for models to compute a soft alignment between the encoder representations and the decoder state at each time-step, through an *attention* mechanism. The computed alignment conditions the decoder more directly on a relevant subset of the input sequence. Computationally, the attention mechanism is typically a simple learned function of the decoder's internal state, e.g., an MLP.

In this work, we propose to augment the encoder-decoder model with attention by integrating a planning mechanism. Specifically, we develop a model that uses planning to

improve the alignment between input and output sequences. It creates an explicit plan of input-output alignments to use at future time-steps, based on its current observation and a summary of its past actions, which it may follow or modify. This enables the model to plan ahead rather than attending to what is relevant primarily at the current generation step. Concretely, we augment the decoder's internal state with (i) an *alignment plan* matrix and (ii) a *commitment plan* vector. The alignment plan matrix is a template of alignments that the model intends to follow at future time-steps, i.e., a sequence of probability distributions over input tokens. The commitment plan vector governs whether to follow the alignment plan at the current step or to recompute it, and thus models discrete decisions. This is reminiscent of macro-actions and options from the hierarchical reinforcement learning literature [**Dietterich, 2000**]. Our planning mechanism is inspired by the *strategic attentive reader and writer* (STRAW) of[**Vezhnevets *et al.*, 2016**], which was originally proposed as a hierarchical reinforcement learning algorithm. In reinforcement-learning parlance, existing sequence-to-sequence models with attention can be said to learn reactive policies ; however, a model with a planning mechanism could learn more proactive policies.

Our work is motivated by the intuition that, although many natural sequences are *output* step-by-step because of constraints on the output process, they are not necessarily *conceived* and *ordered* according to only local, step-by-step interactions. Natural language in the form of speech and writing is again a prime example – sentences are not conceived one word at a time. Planning, that is, choosing some goal along with candidate macro-actions to arrive at it, is one way to induce *coherence* in sequential outputs like language. Learning to generate long coherent sequences, or how to form alignments over long input contexts, is difficult for existing models. In the case of neural machine translation (NMT), the performance of encoder-decoder models with attention deteriorates as sequence length increases [**Cho *et al.*, 2014a, Sutskever *et al.*, 2014b**]. A planning mechanism could make the decoder's search for alignments more tractable and more scalable.

In this work, we perform planning over the input sequence by searching for alignments ; our model does not form an explicit plan of the output tokens to generate. Nevertheless, we find this alignment-based planning to improve performance significantly in several tasks, including character-level NMT. Planning can also be applied explicitly to generation in

sequence-to-sequence tasks. For example, recent work by[**Bahdanau *et al.*, 2016**] on actor-critic methods for sequence prediction can be seen as this kind of generative planning.

We evaluate our model and report results on character-level translation tasks from WMT'15 for English to German, English to Finnish, and English to Czech language pairs. On almost all pairs we observe improvements over a baseline that represents the state-of-the-art in neural character-level translation. In our NMT experiments, our model outperforms the baseline despite using significantly fewer parameters and converges faster in training. We also show that our model performs better than strong baselines on the algorithmic task of finding Eulerian circuits in random graphs and the task of natural-language question generation from a document and target answer.

1.3. Related Works

Existing sequence-to-sequence models with attention have focused on generating the target sequence by aligning each generated output token to another token in the input sequence. This approach has proven successful in neural machine translation [**Bahdanau *et al.*, 2016**] and has recently been adapted to several other applications, including speech recognition [**Chan *et al.*, 2015**] and image caption generation [**Xu *et al.*, 2015**]. In general these models construct alignments using a simple MLP that conditions on the decoder's internal state. In our work we integrate a planning mechanism into the alignment function.

There have been several earlier proposals for different alignment mechanisms : for instance,[**Yang *et al.*, 2016**] developed a hierarchical attention mechanism to perform document-level classification, while [**Luo *et al.*, 2016**] proposed an algorithm for learning discrete alignments between two sequences using policy gradients[**Williams, 1992b**].

[**Silver *et al.*, 2016**] used a planning mechanism based on Monte Carlo tree search with neural networks to train reinforcement learning (RL) agents on the game of Go. Most similar to our work, [**Vezhnevets *et al.*, 2016**] developed a neural planning mechanism, called the strategic attentive reader and writer (STRAW), that can learn high-level temporally abstracted macro-actions. STRAW uses an action plan matrix, which represents the sequences of actions the model plans to take, and a commitment plan vector, which determines whether to commit an action or recompute the plan. STRAW's action plan and commitment plan are stochastic and the model is trained with RL. Our model computes an alignment plan rather

than an action plan, and both its alignment matrix and commitment vector are deterministic and end-to-end trainable with backpropagation.

Our experiments focus on character-level neural machine translation because learning alignments for long sequences is difficult for existing models. This effect can be more pronounced in character-level NMT, since sequences of characters are longer than corresponding sequences of words. Furthermore, to learn a proper alignment between sequences a model often must learn to segment them correctly, a process suited to planning. Previously, [**Chung *et al.*, 2016**] and[**Lee *et al.*, 2016**] addressed the character-level machine translation problem with architectural modifications to the encoder and the decoder. Our model is the first we are aware of to tackle the problem through planning.

1.4. Planning for Sequence-to-Sequence Learning

We now describe how to integrate a planning mechanism into a sequence-to-sequence architecture with attention [**Bahdanau *et al.*, 2015**]. Our model first creates a *plan*, then computes a soft *alignment* based on the plan, and *generates* at each time-step in the decoder. We refer to our model as PAG (Plan-Attend-Generate).

1.4.1. Notation and Encoder

As input our model receives a sequence of tokens, $X = (x_0, \cdots, x_{|X|})$, where $|X|$ denotes the length of X. It processes these with the encoder, a bidirectional RNN. At each input position i we obtain annotation vector $\mathbf{h}_i$ by concatenating the forward and backward encoder states, $\mathbf{h}_i = [\mathbf{h}_i^{\rightarrow}; \mathbf{h}_i^{\leftarrow}]$, where $\mathbf{h}_i^{\rightarrow}$ denotes the hidden state of the encoder's forward RNN and $\mathbf{h}_i^{\leftarrow}$ denotes the hidden state of the encoder's backward RNN.

Through the decoder the model predicts a sequence of output tokens, $Y = (y_1, \cdots, y_{|Y|})$. We denote by $\mathbf{s}_t$ the hidden state of the decoder RNN generating the target output token at time-step t.

1.4.2. Alignment and Decoder

Our goal is a mechanism that plans which parts of the input sequence to focus on for the next k time-steps of decoding. For this purpose, our model computes an alignment plan matrix $\mathbf{A}_t \in \mathbb{R}^{k \times |X|}$ and commitment plan vector $\mathbf{c}_t \in \mathbb{R}^k$ at each time-step. Matrix $\mathbf{A}_t$ stores the alignments for the current and the next $k-1$ timesteps; it is conditioned on the

current input, i.e. the token predicted at the previous time-step, $\mathbf{y}_t$, and the current context $\boldsymbol{\psi}_t$, which is computed from the input annotations $\mathbf{h}_i$. Each row of $\mathbf{A}_t$ gives the logits for a probability vector over the input annotation vectors. The first row gives the logits for the current time-step, t, the second row for the next time-step, $t+1$, and so on. The recurrent decoder function, $f_{\text{dec-rnn}}(\cdot)$, receives $\mathbf{s}_{t-1}$, $\mathbf{y}_t$, $\boldsymbol{\psi}_t$ as inputs and computes the hidden state vector

$$\mathbf{s}_t = f_{\text{dec-rnn}}(\mathbf{s}_{t-1}, \mathbf{y}_t, \boldsymbol{\psi}_t). \tag{1.4.1}$$

Context ψ_t is obtained by a weighted sum of the encoder annotations,

$$\boldsymbol{\psi}_t = \sum_{i}^{|X|} \alpha_{ti} \mathbf{h}_i, \tag{1.4.2}$$

where the soft-alignment vector $\alpha_t = \texttt{softmax}(\mathbf{A}_t[0]) \in \mathbb{R}^{|X|}$ is a function of the first row of the alignment matrix. At each time-step, we compute a candidate alignment-plan matrix $\bar{\mathbf{A}}_t$ whose entry at the i^{th} row is

$$\bar{\mathbf{A}}_t[i] = f_{\text{align}}(\mathbf{s}_{t-1},\, \mathbf{h}_j,\, \boldsymbol{\beta}_t^i,\, \mathbf{y}_t), \tag{1.4.3}$$

where $f_{\text{align}}(\cdot)$ is an MLP and β_t^i denotes a summary of the alignment matrix's i^{th} row at time $t-1$. The summary is computed using an MLP, $f_r(\cdot)$, operating row-wise on $\mathbf{A}_{t-1}$: $\boldsymbol{\beta}_t^i = f_r(\mathbf{A}_{t-1}[i])$.

The commitment plan vector $\mathbf{c}_t$ governs whether to follow the existing alignment plan, by shifting it forward from $t-1$, or to recompute it. Thus, $\mathbf{c}_t$ represents a discrete decision. For the model to operate discretely, we use the recently proposed Gumbel-Softmax trick [**Jang *et al.*, 2016, Maddison *et al.*, 2016**] in conjunction with the straight-through estimator [**Bengio *et al.*, 2013b**] to backpropagate through $\mathbf{c}_t$ [2]. The model further learns the temperature for the Gumbel-Softmax as proposed in [**Gulcehre *et al.*, 2017**]. Both the commitment vector and the action plan matrix are initialized with ones ; this initialization is not modified through training.

Alignment-plan update Our decoder updates its alignment plan as governed by the commitment plan. We denote by g_t the first element of the discretized commitment plan $\bar{\mathbf{c}}_t$. In more detail, $g_t = \bar{\mathbf{c}}_t[0]$, where the discretized commitment plan is obtained by setting

2. We also experimented with training $\mathbf{c}_t$ using REINFORCE [**Williams, 1992b**] but found that Gumbel-Softmax led to better performance.

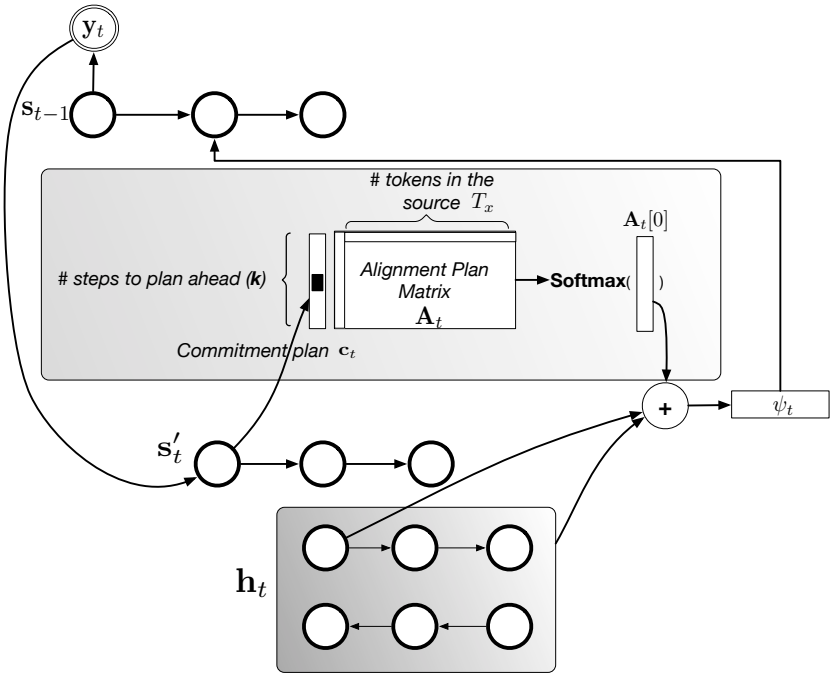


FIGURE 1.1. Our planning mechanism in a sequence-to-sequence model that learns to plan and execute alignments. Distinct from a standard sequence-to-sequence model with attention, rather than using a simple MLP to predict alignments our model makes a plan of future alignments using its alignment-plan matrix and decides when to follow the plan by learning a separate commitment vector. We illustrate the model for a decoder with two layers $\mathbf{s}'_t$ for the first layer and the $\mathbf{s}_t$ for the second layer of the decoder. The planning mechanism is conditioned on the first layer of the decoder ($\mathbf{s}'_t$).

$\mathbf{c}_t$'s largest element to 1 and all other elements to 0. Thus, g_t is a binary indicator variable; we refer to it as the commitment switch. When $g_t = 0$, the decoder simply advances the time index by shifting the action plan matrix $\mathbf{A}_{t-1}$ forward via the shift function $\rho(\cdot)$. When $g_t = 1$, the controller reads the action-plan matrix to produce the summary of the plan, $\boldsymbol{\beta}^i_t$. We then compute the updated alignment plan by interpolating the previous alignment plan matrix $\mathbf{A}_{t-1}$ with the candidate alignment plan matrix $\bar{\mathbf{A}}_t$. The mixing ratio is determined by a learned update gate $\mathbf{u}_t \in \mathbb{R}^{k\times|X|}$, whose elements $\mathbf{u}_{ti}$ correspond to tokens in the input sequence and are computed by an MLP with sigmoid activation, $f_{\text{up}}(\cdot)$:

$$\mathbf{u}_{ti} = f_{\text{up}}(\mathbf{h}_i, \mathbf{s}_{t-1}),$$

$$\mathbf{A}_t[:, i] = (1 - \mathbf{u}_{ti}) \odot \mathbf{A}_{t-1}[:, i] + \mathbf{u}_{ti} \odot \bar{\mathbf{A}}_t[:, i].$$

To reiterate, the model only updates its alignment plan when the current commitment switch g_t is active. Otherwise it uses the alignments planned and committed at previous time-steps.

for $j \in \{1, \cdots |X|\}$ **do**
  for $t \in \{1, \cdots |Y|\}$ **do**
    if $g_t = 1$ **then**
      $\mathbf{c}_t = \text{softmax}(f_c(\mathbf{s}_{t-1}))$
      $\boldsymbol{\beta}_t^j = f_r(\mathbf{A}_{t-1}[j])$ {Read alignment plan}
      $\bar{\mathbf{A}}_t[i] = f_{\text{align}}(\mathbf{s}_{t-1}, \mathbf{h}_j, \boldsymbol{\beta}_t^j, \mathbf{y}_t)$ {Compute candidate alignment plan}
      $\mathbf{u}_{tj} = f_{\text{up}}(\mathbf{h}_j, \mathbf{s}_{t-1}, \boldsymbol{\psi}_{t-1})$ {Compute update gate}
      $\mathbf{A}_t = (1 - \mathbf{u}_{tj}) \odot \mathbf{A}_{t-1} + \mathbf{u}_{tj} \odot \bar{\mathbf{A}}_t$ {Update alignment plan}
    else
      $\mathbf{A}_t = \rho(\mathbf{A}_{t-1})$ {Shift alignment plan}
      $\mathbf{c}_t = \rho(\mathbf{c}_{t-1})$ {Shift commitment plan}
    end if
    Compute the alignment as $\alpha_t = \texttt{softmax}(\mathbf{A}_t[0])$
  end for
end for

Algorithm 3: Pseudocode for updating the alignment plan and commitment vector.

Commitment-plan update The commitment plan also updates when g_t becomes 1. If g_t is 0, the shift function $\rho(\cdot)$ shifts the commitment vector forward and appends a 0-element. If g_t is 1, the model recomputes $\mathbf{c}_t$ using a single layer MLP, $f_c(\cdot)$, followed by a Gumbel-Softmax, and $\bar{\mathbf{c}}_t$ is recomputed by discretizing $\mathbf{c}_t$ as a one-hot vector :

$$\mathbf{c}_t = \texttt{gumbel\_softmax}(f_c(\mathbf{s}_{t-1})), \tag{1.4.4}$$

$$\bar{\mathbf{c}}_t = \texttt{one\_hot}(\mathbf{c}_t). \tag{1.4.5}$$

We provide pseudocode for the algorithm to compute the commitment plan vector and the action plan matrix in Algorithm 3. An overview of the model is depicted in Figure 1.1.

1.4.2.1. *Alignment Repeat*

In order to reduce the model's computational cost, we also propose an alternative to computing the candidate alignment-plan matrix at every step. Specifically, we propose a model variant that reuses the alignment *vector* from the previous time-step until the commitment switch activates, at which time the model computes a new alignment vector. We call this variant *repeat, plan, attend, and generate* (rPAG). rPAG can be viewed as learning an explicit segmentation with an implicit planning mechanism in an unsupervised fashion. Repetition can reduce the computational complexity of the alignment mechanism drastically ; it also eliminates the need for an explicit alignment-plan matrix, which reduces the model's memory consumption also. We provide pseudocode for rPAG in Algorithm 4.

for $j \in \{1, \cdots |X|\}$ **do**
  for $t \in \{1, \cdots |Y|\}$ **do**
    if $g_t = 1$ **then**
      $\mathbf{c}_t = \texttt{softmax}(f_c(\mathbf{s}_{t-1}, \boldsymbol{\psi}_{t-1}))$
      $\boldsymbol{\alpha}_t = \texttt{softmax}(f_{\text{align}}(\mathbf{s}_{t-1}, \mathbf{h}_j, \mathbf{y}_t))$
    else
      $\mathbf{c}_t = \rho(\mathbf{c}_{t-1})$ {Shift the commitment vector $\mathbf{c}_{t-1}$}
      $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}$ {Reuse the old the alignment}
    end if
  end for
end for

Algorithm 4: Pseudocode for updating the repeat alignment and commitment vector.

1.4.3. Training

We use a deep output layer[**Pascanu *et al.*, 2013a**] to compute the conditional distribution over output tokens,

$$p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x}) \propto \mathbf{y}_t^\top \exp(\mathbf{W}_o f_o(\mathbf{s}_t, \mathbf{y}_{t-1}, \boldsymbol{\psi}_t)), \tag{1.4.6}$$

where $\mathbf{W}_o$ is a matrix of learned parameters and we have omitted the bias for brevity. Function f_o is an MLP with tanh activation.

The full model, including both the encoder and decoder, is jointly trained to minimize the (conditional) negative log-likelihood

$$\mathcal{L} = -\frac{1}{N} \sum_{n=1}^{N} \log\ p_\theta(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}),$$

where the training corpus is a set of $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ pairs and θ denotes the set of all tunable parameters. As noted by [**Vezhnevets *et al.*, 2016**], the proposed model can learn to recompute very often, which decreases the utility of planning. To prevent this behavior, we introduce a loss that penalizes the model for committing too often,

$$\mathcal{L}_{\text{com}} = \lambda_{\text{com}} \sum_{t=1}^{|X|} \sum_{i=0}^{k} ||\frac{1}{k} - \mathbf{c}_{ti}||_2^2, \tag{1.4.7}$$

where λ_{com} is the commitment hyperparameter and k is the timescale over which plans operate.

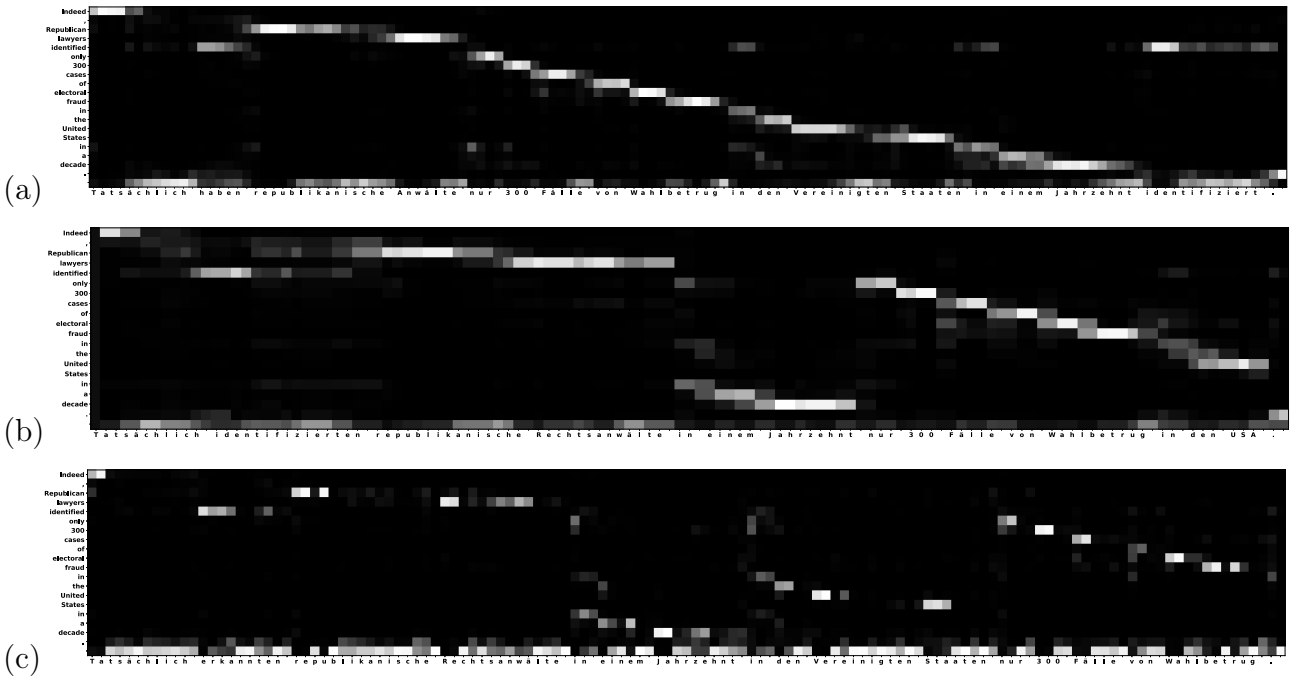


FIGURE 1.2. We visualize the alignments learned by PAG in (a), rPAG in (b), and our baseline model with a 2-layer GRU decoder using $\mathbf{s}_2$ for the attention in (c). As depicted, the alignments learned by PAG and rPAG are smoother than those of the baseline. The baseline tends to put too much attention on the last token of the sequence, defaulting to this empty location in alternation with more relevant locations. Our model, however, places higher weight on the last token usually when no other good alignments exist. We observe that rPAG tends to generate less monotonic alignments in general.

1.5. Experiments

Our baseline is the encoder-decoder architecture with attention described in[**Chung *et al.*, 2016**], wherein the MLP that constructs alignments conditions on the second-layer hidden states, $\mathbf{h}^2$, in the two-layer decoder. The integration of our planning mechanism is analogous across the family of attentive encoder-decoder models, thus our approach can be applied more generally. In all experiments below, we use the same architecture for our baseline and the (r)PAG models. The only factor of variation is the planning mechanism. For training all models we use the Adam optimizer with initial learning rate set to 0.0002. We clip gradients with a threshold of 5[**Pascanu *et al.*, 2013b**] and set the number of planning steps (k) to 10 throughout. In order to backpropagate through the alignment-plan matrices and the commitment vectors, the model must maintain these in memory, increasing the computational overhead of the PAG model. However, rPAG does not suffer from these computational issues.

1.5.1. Algorithmic Task

We first compared our models on the algorithmic task from [**Li *et al.*, 2015**] of finding the "Eulerian Circuits" in a random graph. The original work used random graphs with 4 nodes only, but we found that both our baseline and the PAG model solve this task very easily. We therefore increased the number of nodes to 7. We tested the baseline described above with hidden-state dimension of 360, and the same model augmented with our planning mechanism. The PAG model solves the Eulerian Circuits problem with 100% absolute accuracy on the test set, indicating that for all test-set graphs, all nodes of the circuit were predicted correctly. The baseline encoder-decoder architecture with attention performs well but significantly worse, achieving 90.4% accuracy on the test set.

1.5.2. Question Generation

SQUAD[**Rajpurkar *et al.*, 2016b**] is a question answering (QA) corpus wherein each sample is a (document, question, answer) triple. The document and the question are given in words and the answer is a span of word positions in the document. We evaluate our planning models on the recently proposed question-generation task[**Yuan *et al.*, 2017**], where the goal is to generate a question conditioned on a document and an answer. We add the planning mechanism to the encoder-decoder architecture proposed by [**Yuan *et al.*, 2017**]. Both the document and the answer are encoded via recurrent neural networks, and the model learns to align the question output with the document during decoding. The pointer-softmax mechanism[**Gulcehre *et al.*, 2016**] is used to generate question words from either a short-list vocabulary or by copying from the document. Pointer-softmax uses the alignments to predict the location of the word to copy ; thus, the planning mechanism has a direct influence on the decoder's predictions.

We used 2000 examples from SQUAD's training set for validation and used the official development set as a test set to evaluate our models. We trained a model with 800 units for all GRU hidden states 600 units for word embedding. On the test set the baseline achieved 66.25 NLL while PAG got 65.45 NLL. We show the validation-set learning curves of both models in Figure 1.3.

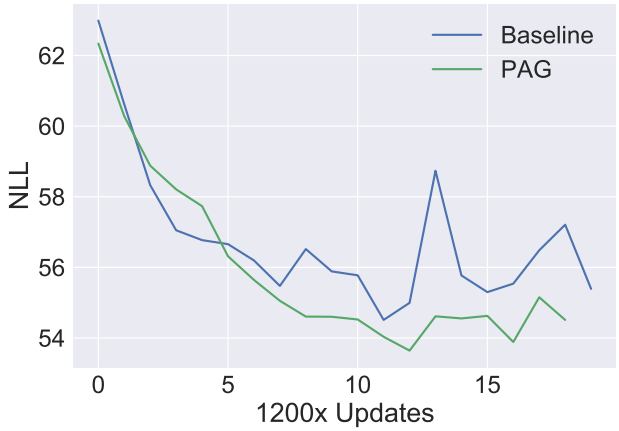


FIGURE 1.3. Learning curves for question-generation models on our development set. Both models have the same capacity and are trained with the same hyperparameters. PAG converges faster than the baseline with better stability.

1.5.3. Character-level Neural Machine Translation

Character-level neural machine translation (NMT) is an attractive research problem [**Lee *et al.*, 2016, Chung *et al.*, 2016, Luong et Manning, 2016**] because it addresses important issues encountered in word-level NMT. Word-level NMT systems can suffer from problems with rare words[**Gulcehre *et al.*, 2016**] or data sparsity, and the existence of compound words without explicit segmentation in some language pairs can make learning alignments between different languages and translations more difficult. Character-level neural machine translation mitigates these issues.

In our NMT experiments we use byte pair encoding (BPE)[**Sennrich *et al.*, 2015**] for the source sequence and characters at the target, the same setup described in[**Chung *et al.*, 2016**]. We also use the same preprocessing as in that work.[3] We present our experimental results in Table 1.1. Models were tested on the WMT'15 tasks for English to German (En→De), English to Czech (En→Cs), and English to Finnish (En→Fi) language pairs. The table shows that our planning mechanism improves translation performance over our baseline (which reproduces the results reported in [**Chung *et al.*, 2016**] to within a small margin). It does this with fewer updates and fewer parameters. We trained (r)PAG for 350K updates on the training set, while the baseline was trained for 680K updates. We used 600 units in (r)PAG's encoder and decoder, while the baseline used 512 in the encoder and 1024 units in the decoder. In total our model has about 4M fewer parameters than the baseline. We tested all models with a beam size of 15.

3. Our implementation is based on the code available at `https://github.com/nyu-dl/dl4mt-cdec`

As can be seen from Table 1.1, layer normalization [**Ba *et al.*, 2016**] improves the performance of PAG significantly. However, according to our results on En→De, layer norm affects the performance of rPAG only marginally. Thus, we decided not to train rPAG with layer norm on other language pairs.

	Model	Layer Norm	Dev	Test 2014	Test 2015
En→De	Baseline	✗	21.57	21.33	**23.45**
	Baseline†	✗	21.4	21.16	22.1
	Baseline†	✓	21.65	21.69	22.55
	PAG	✗	21.92	21.93	22.42
		✓	**22.44**	**22.59**	**23.18**
	rPAG	✗	21.98	22.17	22.85
		✓	22.33	22.35	22.83
En→Cs	Baseline	✗	17.68	19.27	16.98
	Baseline†	✓	19.1	21.35	18.79
	PAG	✗	18.9	20.6	18.88
		✓	**19.44**	**21.64**	**19.48**
	rPAG	✗	18.66	21.18	19.14
En→Fi	Baseline	✗	11.19	-	10.93
	Baseline†	✓	11.26	-	10.71
	PAG	✗	12.09	-	11.08
		✓	**12.85**	-	**12.15**
	rPAG	✗	11.76	-	11.02

Table 1.1. The results of different models on the WMT'15 tasks for English to German, English to Czech, and English to Finnish language pairs. We report BLEU scores of each model computed via the *multi-blue.perl* script. The best-score of each model for each language pair appears in bold-face. We use *newstest2013* as our development set, *newstest2014* as our "Test 2014" and *newstest2015* as our "Test 2015" set. $(^{\dagger})$ denotes the results of the baseline that we trained using the hyperparameters reported in [**Chung *et al.*, 2016**] and the code provided with that paper. For our baseline, we only report the median result, and do not have multiple runs of our models. On WMT'14 and WMT'15 for En*rightarrow*De character-level NMT, [**Kalchbrenner *et al.*, 2016**] have reported better results with deeper auto-regressive convolutional models (Bytenets), 23.75 and 26.26 respectively.

In Figure 1.2, we show qualitatively that our model constructs smoother alignments. At each word that the baseline decoder generates, it aligns the first few characters to a word in the source sequence, but for the remaining characters places the largest alignment weight on the last, empty token of the source sequence. This is because the baseline becomes confident of which word to generate after the first few characters, and it generates the remainder of the word mainly by relying on language-model predictions. We observe that (r)PAG converges

faster with the help of the improved alignments, as illustrated by the learning curves in Figure 1.4.

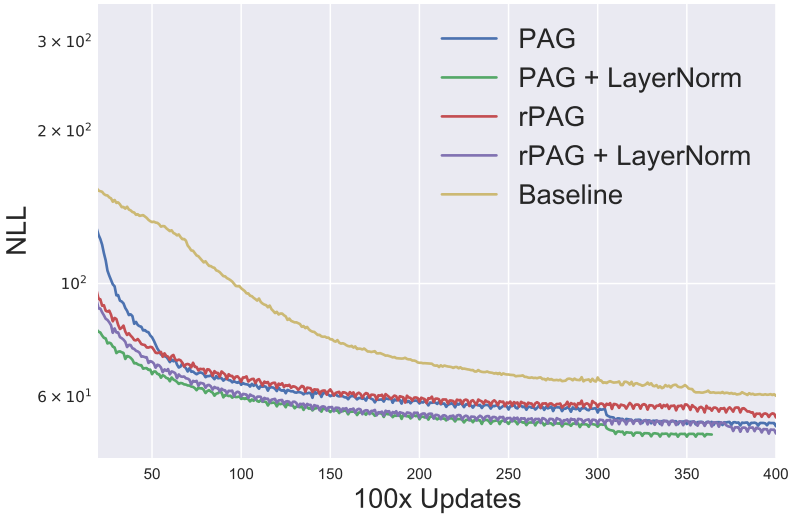


Figure 1.4. Learning curves for different models on WMT'15 for En→De. Models with the planning mechanism converge faster than our baseline (which has larger capacity).

1.6. Conclusion

In this work we addressed a fundamental issue in neural generation of long sequences by integrating *planning* into the alignment mechanism of sequence-to-sequence architectures. We proposed two different planning mechanisms : PAG, which constructs explicit plans in the form of stored matrices, and rPAG, which plans implicitly and is computationally cheaper. The (r)PAG approach empirically improves alignments over long input sequences. We demonstrated our models' capabilities through results on character-level machine translation, an algorithmic task, and question generation. In machine translation, models with planning outperform a state-of-the-art baseline on almost all language pairs using fewer parameters. We also showed that our model outperforms baselines with the same architecture (minus planning) on question-generation and algorithmic tasks. The introduction of planning improves training convergence and potentially the speed by using the alignment repeats.

Chapitre 2

Génération de langue naturelle à l'aide de modèles génératifs

Adversarial Generation of Natural Language

Sai Rajeswar*, Sandeep Subramanian*, Francis Dutil, Christopher Pal, Aaron Courville [1]

Contribution personelle Pour cet article, j'ai principalement participé à la partie implémentation. J'ai implémenté notamment la partie $WGAN - GP$ et mis en place le code pour le pipeline d'évaluation, d'entraînement et chargement des modèles. J'ai également aidé à rouler plusieurs expériences pour la tâche de génération de langue, présentée dans la section 2.5.4.

Affiliations

Sai Rajeswar*, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Sandeep Subramanian*, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Francis Dutil, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Christopher Pal, MILA, Ecole Polytechnique de Montréal

Aaron Courville, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CIFAR Fellow

1. * Signifie une contribution égale au projet

2.1. Abstract

Generative Adversarial Networks (GANs) have gathered a lot of attention from the computer vision community, yielding impressive results for image generation. Advances in the adversarial generation of natural language from noise however are not commensurate with the progress made in generating images, and still lag far behind likelihood based methods. In this paper, we take a step towards generating natural language with a GAN objective alone. We introduce a simple baseline that addresses the discrete output space problem without relying on gradient estimators and show that it is able to achieve state-of-the-art results on a Chinese poem generation dataset. We present quantitative results on generating sentences from context-free and probabilistic context-free grammars, and qualitative language modeling results. A conditional version is also described that can generate sequences conditioned on sentence characteristics.

2.2. Introduction

Deep neural networks have recently enjoyed some success at modeling natural language [**Mikolov *et al.*, 2010, Zaremba *et al.*, 2014, Kim *et al.*, 2015**]. Typically, recurrent and convolutional language models are trained to maximize the likelihood of observing a word or character given the previous observations in the sequence $P(w_1 \dots w_n) = p(w_1) \prod_{i=2}^{n} P(w_i | w_1 \dots w_{i-1})$. These models are commonly trained using a technique called *teacher forcing* [**Williams et Zipser, 1989**] where the inputs to the network are fixed and the model is trained to predict only the next item in the sequence given all previous observations. This corresponds to maximum-likelihood training of these models. However this one-step ahead prediction during training makes the model prone to *exposure bias* [**Ranzato *et al.*, 2016, Bengio *et al.*, 2015**]. Exposure bias occurs when a model is only trained conditioned on ground-truth contexts and is not exposed to its own errors [**Wiseman et Rush, 2016**]. An important consequence to exposure bias is that generated sequences can degenerate as small errors accumulate. Many important problems in NLP such as machine translation and abstractive summarization are trained via a maximum-likelihood training objective [**Bahdanau *et al.*, 2014b, Rush *et al.*, 2015**], but require the generation of extended sequences and are evaluated based on sequence-level metrics such as BLEU [**Papineni *et al.*, 2002**] and ROUGE [**Lin, 2004**].

One possible direction towards incorporating a sequence-level training objective is to use Generative Adversarial Networks (GANs) [**Goodfellow *et al.*, 2014**]. While GANs have yielded impressive results for modeling images [**Radford *et al.*, 2015, Dumoulin *et al.*, 2016**], advances in their use for natural language generation has lagged behind. Some progress has been made recently in incorporating a GAN objective in sequence modeling problems including natural language generation. [**Lamb *et al.*, 2016**] use an adversarial criterion to match the hidden state dynamics of a teacher forced recurrent neural network (RNN) and one that samples from its own output distribution across multiple time steps. Unlike the approach in [**Lamb *et al.*, 2016**], sequence GANs [**Yu *et al.*, 2016**] and maximum-likelihood augmented GANs [**Che *et al.*, 2017**] use an adversarial loss at outputs of an RNN. Using a GAN at the outputs of an RNN however isn't trivial since sampling from these discrete outputs to feed to the discriminator is a non-differentiable operation. As a result gradients cannot propagate to the generator from the discriminator. [**Yu *et al.*, 2016**] use policy gradient to estimate the generator's gradient and [**Che *et al.*, 2017**] present an importance sampling based technique. Other alternatives include REINFORCE [**Williams, 1992a**], the use of a Gumbel softmax [**Jang *et al.*, 2016**] and the straighthrough estimator [**Bengio *et al.*, 2013a**] among others.

In this work, we address the discrete output space by simply forcing the discriminator to operate on continuous valued output distributions. The discriminator sees a sequence of probabilities over every token in the vocabulary from the generator and a sequence of 1-hot vectors from the true data distribution as in Fig. 2.1. This technique is identical to that proposed by [**Gulrajani *et al.*, 2017**], which is parallel work to this. In this paper we provide a more complete empirical investigation of this approach to applying GANs to discrete output spaces. We present results using recurrent as well as convolutional architectures on three language modeling datasets of different sizes at the word and character-level. We also present quantitative results on generating sentences that adhere to a simple context-free grammar (CFG), and a richer probabilistic context-free grammar (PCFG). We compare our method to previous works that use a GAN objective to generate natural language, on a Chinese poetry generation dataset. In addtion, we present a conditional GAN [**Mirza et Osindero, 2014**] that generates sentences conditioned on sentiment and questions.

2.3. Generative Adversarial Networks

GANs [**Goodfellow *et al.*, 2014**] are a general framework used in training generative models by formulating the learning process as a two player minimax game as formulated in the equation below. A generator network G tries to generate samples that are as close as possible to the true data distribution $P(x)$ of interest from a fixed noise distribution $P(z)$. We will refer to the samples produced by the generator as $G(z)$. A discriminator network is then trained to distinguish between $G(z)$ and samples from the true data distribution $P(x)$ while the generator network is trained using gradient signals sent by the discriminator by minimizing $\log(1 - D(G(z)))$. [**Goodfellow *et al.*, 2014**] have shown that, with respect to an optimal discriminator, the minimax formulation can be shown to minimize the Jensen Shannon Divergence (JSD) between the generator's output distribution and the true data distribution.

$$\min_G \max_D V(D, G) = \mathop{\mathbb{E}}_{x \sim P(x)} [\log D(x)] + \mathop{\mathbb{E}}_{z \sim P(z)} [\log(1 - D(G(z)))]$$

However, in practice, the generator is trained to maximize $\log(D(G(z)))$ instead, since it provides stronger gradients in the early stages of learning [**Goodfellow *et al.*, 2014**].

GANs have been reported to be notoriously hard to train in practice [**Arjovsky et Bottou, 2017**] and several techniques have been proposed to alleviate some of the complexities involved in getting them to work including modified objective functions and regularization [**Salimans *et al.*, 2016, Arjovsky *et al.*, 2017, Mao *et al.*, 2016, Gulrajani *et al.*, 2017**]. We demonstrate the importance choice of cost function and regularization is critical and can significantly affect model performance.

[**Nowozin *et al.*, 2016**] show that it is possible to train GANs with a variety of f-divergence measures besides JSD. Wasserstein GANs (WGANs) [**Arjovsky *et al.*, 2017**] minimize the earth mover's distance or Wasserstein distance, while Least Squared GANs (LSGANs) [**Mao *et al.*, 2016**] replaces the log loss with an L2 loss. WGAN-GP [**Gulrajani *et al.*, 2017**] incorporate a gradient penalty term on the discriminator's loss in the WGAN objective which acts as a regularizer. In this work, we will compare some of these objectives in the context of natural language generation.

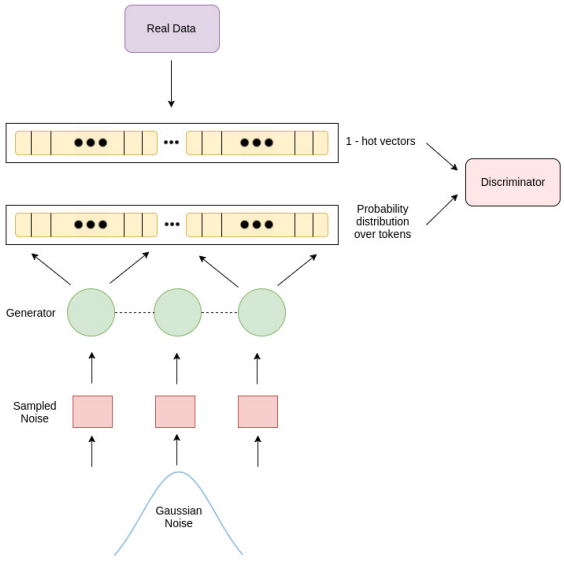


FIGURE 2.1. Model architecture

2.4. Model architecture

In this section, we will present the details of our model including choices of neural architectures for our generator, discriminator and training objectives.

Let $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ be the input to our generator network G from which we will attempt to generate natural language. For implementation convenience, the sample $\mathbf{z}$ is of shape $n \times d$ where n is the length of sequence and d is a fixed length dimension of the noise vector at each time step. The generator then transforms $\mathbf{z}$ into a sequence of probability distributions over the vocabulary $G(z)$ of size $n \times k$ where k is the size of our true data distribution's vocabulary. The discriminator network D is provided with fake samples $G(z)$ and samples from the true data distribution $P(x)$. Samples from the true distribution are provided as a sequence of 1-hot vectors with each vector serving as an indicator of the observed word in the sample. As described in section 2, the discriminator is trained to discriminate between real and fake samples and the generator is trained to fool the discriminator as in Fig. 2.1.

We investigate recurrent architectures as in [**Lamb *et al.*, 2016, Yu *et al.*, 2016, Che *et al.*, 2017**] and convolutional architectures in both the generator as well as the discriminator. The following subsections detail our architectures.

2.4.1. Recurrent Models

Recurrent Neural Networks (RNNs), particularly Long short-term memory networks (LSTMs) [**Hochreiter et Schmidhuber, 1997**] and Gated Recurrent Networks

[**Cho *et al.*, 2014b**] are powerful autoregressive models that have been successful at modeling sequential data [**Graves et Schmidhuber, 2009, Mikolov *et al.*, 2010**]. They transform a sequence of input vectors $\mathbf{x} = x_1 \dots x_n$ into a sequence of hidden states $\mathbf{h} = h_1 \dots h_n$ where each hidden state maintains a summary of the input up until then. RNN language models are autoregressive in nature since the input to the network at time t depends on the output at time $t-1$. However, in the context of generating sequences from noise, the inputs are pre-determined and there is no direct correspondence between the output at time $t-1$ and the input at time t this fundamentally changes the auto-regressiveness of the RNN. The RNN does however carry forward information about its output at time t through subsequent time steps via its hidden states $\mathbf{h}$ as evident from the recurrent transition function. In order to incorporate an explicit dependence between subsequent RNN outputs, we add a peephole connection between the *output* probability distribution $\mathbf{y_{t-1}}$ at time $t-1$ and the hidden state $\mathbf{h_t}$ at time t as follows.

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{pi}\mathbf{y}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{pf}\mathbf{y}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{po}\mathbf{y}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}_t &= \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{W}_{pc}\mathbf{y}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}
$$

where σ is the element-wise sigmoid function, $\odot$ is the hadamard product, tanh is the element-wise tanh function. $\mathbf{W}.$ and $\mathbf{b}.$ are learn-able parameters of the model and $\mathbf{i_t}$, $\mathbf{f_t}$, $\mathbf{o_t}$ and $\mathbf{c_t}$ constitute the input, forget, output and cell states of the LSTM respectively. Typical RNN language models have a shared affine transformation matrix $\mathbf{W_{out}}$ that is shared across time all steps that projects the hidden state vector to a vector of the same size as the target vocabulary to generate a sequence of outputs $\mathbf{y} = y_1 \dots y_t$. Subsequently a softmax function is applied to each vector to turn it into a probability distribution over the vocabulary.

$$
\mathbf{y}_t = \text{softmax}(\mathbf{W}_{out}\mathbf{h}_t + \mathbf{b}_{out}),
$$

During inference, an output is sampled from the softmax distribution and becomes the input at the subsequent time step. While training the inputs are pre-determined. In all of our models, we perform greedy decoding where we always pick argmax y_t. When using the LSTM as a discriminator we use a simple binary logistic regression layer on the last hidden state h_n to determine the probability of the sample being from the generator's data distribution or from the real data distribution.

$$P(real) = \sigma(\mathbf{W}_{pred}\mathbf{h}_n + \mathbf{b}_{pred}),$$

2.4.2. Convolutional Models

Convolutional neural networks (CNNs) have also shown promise at modeling sequential data using 1-dimensional convolutions [**Dauphin *et al.*, 2016, Zhang *et al.*, 2015**]. Convolution filters are convolved across time and the input dimensions are treated as channels. In this work, we explore convolutional generators and discriminators with residual connections [**He *et al.*, 2016**].

[**Gulrajani *et al.*, 2017**] use a convolutional model for both the generator and discriminator. The generator consists of 5 residual blocks with 2 1-D convolutional layers each. A final 1-D convolution layer transforms the output of the residual blocks into a sequence of un-normalized vectors for each element in the input sequence (noise). These vectors are then normalized using the softmax function. All convolutions are 'same' convolutions with a stride of 1 followed by the ReLU [**Nair et Hinton, 2010, Glorot *et al.*, 2011**] activation function without any pooling so as to preserve the shape of the input. The discriminator architecture is identical to that of the generator with the final output having a signal output channel. We also experiment with a smaller discriminator architecture with 3 convolution layers with max pooling and batch-normalization [**Ioffe et Szegedy, 2015b**].

2.4.3. Curriculum Learning

In likelihood based training of generative language models, models are only trained to make one-step ahead predictions and as a result it is possible to train these models on relatively long sequences even in the initial stages of training. However, in our adversarial formulation, our generator is encouraged to generate entire sequences that match the true data distribution without explicit supervision at each step of the generation process.

As a way to provide training signals of incremental difficulty, we use curriculum learning [**Bengio *et al.*, 2009**] and train our generator to produce sequences of gradually increasing lengths as training progresses.

2.5. Experiments & Data

In this section, we present our experimental setup, datasets and experimental details.

GAN based methods have often been critiqued for lacking a concrete evaluation strategy [**Salimans *et al.*, 2016**], however recent work [**Wu *et al.*, 2016**] uses an annealed importance based technique to overcome this problem.

In the context of generating natural language, it is possible to come up with a simpler approach to evaluate compute the likelihoods of generated samples. We synthesize a data generating distribution under which we can compute likelihoods in a tractable manner. We propose a simple evaluation strategy for evaluating adversarial methods of generating natural language by constructing a data generating distribution from a CFG or P-CFG. It is possible to determine if a sample belongs to the CFG or the probability of a sample under a P-CFG by using a constituency parser that is provided with all of the productions in a grammar. [**Yu *et al.*, 2016**] also present a simple idea to estimate the likelihood of generated samples by using a randomly initialized LSTM as their data generating distribution. While this is a viable strategy to evaluate generative models of language, a randomly initialized LSTM provides little visibility into the complexity of the data distribution itself and presents no obvious way to increase its complexity. CFGs and PCFGs however, provide explicit control of the complexity via their productions. They can also be learned via grammar induction [**Brill, 1993**] on large treebanks of natural language and so the data generating distribution is not synthetic as in [**Yu *et al.*, 2016**].

We divide our experiments into three categories :

— Generating language that belongs to a toy CFG and an induced PCFG from the Penn Treebank [**Marcus *et al.*, 1993**].

— Chinese poetry generation with comparisons to [**Yu *et al.*, 2016**] and [**Che *et al.*, 2017**].

— Generated samples from a dataset consisting of simple English sentences, the 1-billion-word and Penn Treebank datasets.

— Conditional GANs that generate sentences conditioned on certain sentence attributes such as sentiment and questions.

2.5.1. Simple CFG

We use a simple and publicly available CFG [2] that contains 248 productions. We then generate two sets of data from this CFG - one consisting of samples of length 5 and another of length 11. Each set contains 100,000 samples selected at random from the CFG. The first set has a vocabulary of 36 tokens while the second 45 tokens. We evaluate our models on this task by measuring the fraction of generated samples that satisfy the rules of the grammar and also measure the diversity in our generated samples. We do this by generating 1,280 samples from noise and computing the fraction of those that are valid under our grammar using the Earley parsing algorithm [**Earley, 1970**]. In order to measure sample diversity, we simply use the count the number of unique samples ; while this assumes that all samples are orthogonal it still serves as a proxy measure of the entropy. We compare various generator, discriminator and GAN objectives on this problem.

2.5.2. Penn Treebank PCFG

To construct a more challenging problem than a simple CFG, we use sections 0-21 of the WSJ subsection of the Penn Treebank to induce a PCFG using simple count statistics of all productions.

$$P(A \to BC) = \frac{count(A \to BC)}{count(A \to *)}$$

We train our model on all sentences in the treebank and restrict the output vocabulary to the top 2,000 most frequently occurring words. We evaluate our models on this task by measuring the likelihood of a sample using a Viterbi chart parser [**Klein et Manning, 2003**].

2.5.3. Chinese Poetry

[**Zhang et Lapata, 2014**] present a dataset of Chinese poems that were used to evaluate adversarial training methods for natural language in [**Yu *et al.*, 2016**] and [**Che *et al.*, 2017**]. The dataset consists of 4-line poems with a variable number of characters in each line. We treat each line in a poem as a training example and use lines

2. `http://www.cs.jhu.edu/~jason/465/hw-grammar/extra-grammars/holygrail`

of length 5 (poem-5) and 7 (poem-7) with the train/validation/test split [3] specified in [**Che *et al.*, 2017**]. We use BLEU-2 and BLEU-3 to measure model performance on this task. Since there is no obvious "target" for each generated sentence, both works report corpus-level BLEU measures using the entire test set as the reference. Since there is no obvious method to measure the likelihood of our generated samples, we cannot report model perplexity.

2.5.4. Language Generation

We generate language from three different datasets of varying sizes and complexity. A dataset comprising simple English sentences [4] which we will henceforth refer to as CMU−SE, the version of the Penn Treebank commonly used in language modeling experiments [**Zaremba *et al.*, 2014**] and the Google 1-billion word dataset [**Chelba *et al.*, 2013**]. We perform experiments at generating language at the word as well as character-level. The CMU−SE dataset consists of 44,016 sentences with a vocabulary of 3,122 words while the Penn Treebank consists of 42,068 sentences with a vocabulary of 10,000 words. We use a random subset of 3 million sentences from the 1-billion word dataset with and constrain our vocabulary to the top 30,000 most frequently occurring words. We use a curriculum learning strategy in all of our LSTM models (with and without the output peephole connection) that starts training on sentences of length 5 at the word level and 13 for characters and increases the sequence length by 1 after a fixed number of epochs based on the size of the data. Convolutional methods in [**Gulrajani *et al.*, 2017**] are able to generate long sequences even without a curriculum however we found a curriculum was critical in generating long sequences with an LSTM.

2.5.5. Conditional Generation of Sequences

GANs are able to leverage explicit conditioning on high-level attributes of data [**Mirza et Osindero, 2014, Gauthier, 2014, Radford *et al.*, 2015**] to generate samples which contain these attributes. Recent work [**Hu *et al.*, 2017**] generates sentences conditioned on certain attributes of language such as sentiment using a variational autoencoders (VAEs) [**Kingma et Welling, 2013**] and holistic attribute discriminators.

3. `http://homepages.inf.ed.ac.uk/mlap/Data/EMNLP14/`
4. `https://github.com/clab/sp2016.11-731/tree/master/hw4/data`

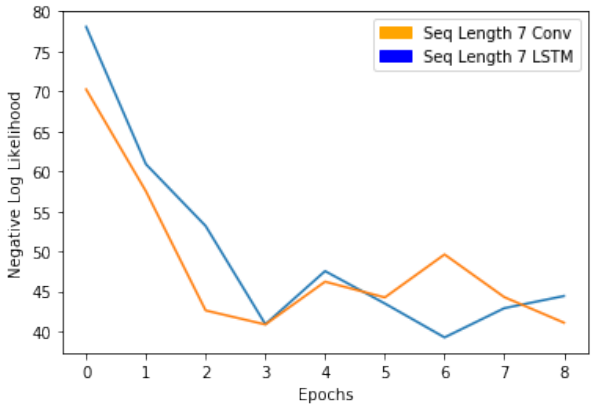


FIGURE 2.2. Negative log-likelihood of generated samples under our PCFG

In this paper, we use two features inherent in language - sentiment and questions. To generate sentences that are questions, we use the CMU-SE dataset and label sentences that contain a " ?" as being questions and the rest as been statements. To generate sentences of positive and negative sentiment we use the Amazon review polarity dataset collected in **[Zhang *et al.*, 2015]** and use the first 3 million *short* reviews with a vocabulary of the top 4,000 most frequently occurring words. Conditioning on sentence attributes is achieved by concatenating a single feature map containing either entirely ones or zeros to indicate the presence or absence of the attribute as in **[Radford *et al.*, 2015]** at the output of each convolutional layer. The conditioning is done on both the generator and the discriminator. We experiment with conditional GANs using only convolutional methods since methods adding conditioning information has been well studied in these architectures.

2.5.6. Training

All models are trained using the back-propagation algorithm updating our parameters using the Adam optimization method **[Kingma et Ba, 2014]** stochastic gradient descent (SGD) with batch sizes of 64 and . A learning rate of 2×10^{-3}, $\beta_1 = 0.5$ and $\beta_2 = 0.999$ is used in our LSTM generator and discriminators while convolutional architectures use a learning rate of 1×10^{-4}. The noise prior $P(z)$ and all LSTM hidden dimensions are set to 128 except for the Chinese poetry generation task where we use a hidden dimension of size 64.

2.6. Results and Discussion

In this section, we present results on the various tasks described in Section 4.

Gen	Disc	Objective	Length 5		Length 11	
			Acc (%)	Uniq	Acc (%)	Uniq
LSTM	LSTM	GAN	99.06	0.913	0	0.855
LSTM	LSTM	LSGAN	99.45	0.520	0	0.855
LSTM	LSTM	WGAN	93.98	0.972	98.04	0.924
LSTM-P	LSTM	WGAN	97.96	0.861	99.29	0.653
LSTM	LSTM	WGAN-GP	99.21	0.996	96.25	0.992
CNN	CNN	WGAN-GP	98.59	0.990	97.01	0.771

Table 2.1. Simple CFG task results

Table. 2.1 presents quantitative results on generating sentences that adhere to the simple CFG described in Section 4.1. The Acc column computes the accuracy with which our model generates samples from the CFG using a sample of 1,280 generations. We observe that all models are able to fit sequences of length 5 but only the WGAN and WGAN-GP objectives are able to generalize to longer sequences of length 11. This motivated us to use only the WGAN and WGAN-GP objectives in our subsequent experiments. GANs have been shown to exhibit the phenomenon of "mode dropping" where the generator fails to capture a large fraction of the modes present in the data generating distribution [**Che *et al.*, 2016**]. It is therefore important to study the diversity in our generated samples. The Uniq column computes the number of unique samples in a sample 1,280 generations serves as a rough indicator of sample diversity. The WGAN-GP objective appears to encourage the generation of diverse samples while also fitting the data distribution well.

Fig. 2.2 shows the negative-log-likelihood of generated samples using a CNN architecture using the WGAN-GP criterion and an LSTM using the WGAN criterion. The sequence length is set to 7 and the likelihoods are evaluated at the end of every epoch on a set of 64 samples. Both models learn to gradually fit the data distribution.

Table. 2.2 contains quantitative results on the Chinese poetry generation dataset. The results indicate that our straightforward strategy to overcome back-propagating through discrete states is competitive and outperforms more complicated methods [**Yu *et al.*, 2016, Che *et al.*, 2017**].

Table. 2.5 contains sequences generated by our model conditioned on sentiment (positive/negative) and questions/statements. The model is able to pick up on certain consistent patterns in questions as well as when expressing sentiment and use them while generating sentences.

Models	Poem 5				Poem 7			
	BLEU-2		BLEU-3		BLEU-2		BLEU-3	
	Val	Test	Val	Test	Val	Test	Val	Test
MLE [**Che *et al.*, 2017**]	-	0.693	-	-	-	0.318	-	-
Sequence GAN [**Yu *et al.*, 2016**]	-	0.738	-	-	-	-	-	-
MaliGAN-basic [**Che *et al.*, 2017**]	-	0.740	-	-	-	0.489	-	-
MaliGAN-full [**Che *et al.*, 2017**]	-	0.762	-	-	-	0.552	-	-
LSTM (ours)	0.840	0.837	0.427	**0.372**	0.660	0.655	0.386	**0.405**
LSTM Peephole (ours)	0.845	**0.878**	0.439	0.363	0.670	**0.670**	0.327	0.355

Table 2.2. BLEU scores on the poem-5 and poem-7 datasets

Tables 2.3 and 2.4 contain sequences generated at the word and character-level by our LSTM and CNN models. Both models are able to produce realistic sentences, but often appear to capture only local syntactic and semantic structure. The CNN model with a WGAN-GP objective appears to be able to maintain context over longer time spans.

Level	Method	1-billion-word
Word	LSTM	An opposition was growing in China . This is undergoing operation a year . It has his everyone on a blame . Everyone shares that Miller seems converted President as Democrat . Which is actually the best of his children . Who has The eventual policy and weak ?
	CNN	Companies I upheld , respectively patented saga and Ambac. Independence Unit have any will MRI in these Lights It is a wrap for the annually of Morocco The town has Registration matched with unk and the citizens
Character	CNN	To holl is now my Hubby , The gry timers was faller After they work is jith a But in a linter a revent

Table 2.3. Word and character-level generations on the 1-billion word dataset

2.7. Conclusion and Future work

In conclusion, this work presents a straightforward but effective method to train GANs for natural language. The simplicity lies in *forcing the discriminator to operate on continuous values* by presenting it with a sequence of probability distributions from the generator and a sequence of 1-hot vectors corresponding to data from the true distribution. We propose an evaluation strategy that involves learning the data distribution defined by a CFG or

Level	Model	PTB	CMU-SE
Word	LSTM	what everything they take everything away from .	<s>will you have two moment ? </s>
		may tea bill is the best chocolate from emergency .	<s>i need to understand deposit length . </s>
		can you show show if any fish left inside .	<s>how is the another headache ? </s>
		room service , have my dinner please .	<s>how there , is the restaurant popular this cheese ? </s>
	CNN	meanwhile henderson said that it has to bounce for.	<s>i 'd like to fax a newspaper . </s>
		I'm at the missouri burning the indexing manufacturing and through .	<s>cruise pay the next in my replacement . </s>
			<s>what 's in the friday food ? ? </s>

Table 2.4. Word level generations on the Penn Treebank and CMU-SE datasets

POSITIVE	NEGATIVE
best and top notch newtonmom .	usuall the review omnium nothing non-functionable
good buy homeostasis money well spent	
kickass cosamin of time and fun .	extreme crap-not working and eeeeeew
great britani ! I lovethis.	a horrible poor imposing se400
QUESTION	STATEMENT
<s>when 's the friday convention on ? </s>	<s>i report my run on one mineral . </s>
<s>how many snatched crew you have ? </s>	<s>we have to record this now . </s>
<s>how can you open this hall ? </s>	<s>i think i deeply take your passenger .</s>

Table 2.5. Conditional generation of text. Top row shows generated samples conditionally trained on amazon review polarity dataset with two attributes 'positive' and 'negative'. Bottom row has samples conditioned on the 'question' attribute

PCFG. This lets us evaluate the likelihood of a sample belonging to the data generating distribution. The use of WGAN and WGAN-GP objectives produce realistic sentences on datasets of varying complexity (CMU-SE, Penn Treebank and the 1-billion dataset). We also show that it is possible to perform conditional generation of text on high-level sentence features such as sentiment and questions. In future work, we would like to explore GANs in other domains of NLP such as machine translation, and non-goal oriented dialog systems where a clear training and evaluation criterion does not exist.

Chapitre 3

Conclusion

Dans ce mémoire, nous avons présenté deux articles sur la génération et prédiction de séquences.

Dans le chapitre 1, nous avons mis au point un système de planification d'attention dans les modèles de séquence-à-séquence. Une décision discrète était prise afin d'établir la prochaine planification. Nous avons montré que l'approche aidait l'apprentissage des alignements et offrait des gains de performances sur diverses tâches.

Dans le futur, nous pourrions explorer différents moyens d'incorporer la planification dans ces réseaux. [**Serdyuk *et al.*, 2017**] par exemple, entraîne deux réseaux. Un *forward* RNN qui traite les données $\mathbf{x}_i$ en ordre chronologique et un *backward* RNN qui les traite en sens inverse. Durant l'entraînement, un terme de régularisation est ajouté pour forcer l'état caché des deux réseaux à s'enligner. Durant la phase de génération, seul le *forward* RNN est utilisé. Nous pouvons voir cette régularisation comme encourageant le *forward* RNN à planifier à l'avance la valeur de son état caché.

Dans le chapitre 2, nous avons travaillé sur la génération de langues naturelles avec des réseaux antagonistes génératifs. Nous avons montré qu'en utilisant un discriminateur adéquat, nous étions en mesure de générer des séquences discrètes, sans aucune estimation de gradients.

Plusieurs directions de recherche restent encore à être explorées en génération de langues naturelles. Nous pouvons par exemple étudier la combinaison de ce type de modèles avec ceux entraînés avec MLE. Ceci pourrait contrebalancer l'effet du *teacher forcing* et aider la prédiction de longues séquences. Différents modèles génératifs peuvent également être explorés. [**Zhao *et al.*, 2018**] par exemple, utilise un *Wasseintein autoencoder* (WAE)

[**Tolstikhin *et al.*, 2017**], dont la distribution à priori est apprise à l'aide d'un GAN. Tout comme le modèle présenté dans le chapitre 2, ceci permet la génération de données discrètes sans aucune estimation de gradients.

Bibliographie

[Arjovsky et Bottou, 2017] Arjovsky, M. et Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *In NIPS 2016 Workshop on Adversarial Training. In review for ICLR*, volume 2016.

[Arjovsky *et al.*, 2017] Arjovsky, M., Chintala, S. et Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv :1701.07875*.

[Ba *et al.*, 2016] Ba, J. L., Kiros, J. R. et Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv :1607.06450*.

[Bahdanau *et al.*, 2016] Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A. et Bengio, Y. (2016). An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv :1607.07086*.

[Bahdanau *et al.*, 2014a] Bahdanau, D., Cho, K. et Bengio, Y. (2014a). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Bahdanau *et al.*, 2014b] Bahdanau, D., Cho, K. et Bengio, Y. (2014b). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv :1409.0473*.

[Bahdanau *et al.*, 2015] Bahdanau, D., Cho, K. et Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*.

[Bengio *et al.*, 2015] Bengio, S., Vinyals, O., Jaitly, N. et Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *In Advances in Neural Information Processing Systems*, pages 1171–1179.

[Bengio, 2013] Bengio, Y. (2013). Deep learning of representations : Looking forward. *In International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer.

[Bengio *et al.*, 2013a] Bengio, Y., Léonard, N. et Courville, A. (2013a). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv :1308.3432*.

[Bengio *et al.*, 2013b] Bengio, Y., Léonard, N. et Courville, A. (2013b). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv :1308.3432*.

[Bengio *et al.*, 2009] Bengio, Y., Louradour, J., Collobert, R. et Weston, J. (2009). Curriculum learning. *In Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

[Bengio *et al.*, 1994] Bengio, Y., Simard, P. et Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Brill, 1993] Brill, E. (1993). Automatic grammar induction and parsing free text : A transformation-based approach. *In Proceedings of the workshop on Human Language Technology*, pages 237–242. Association for Computational Linguistics.

[Chan *et al.*, 2015] Chan, W., Jaitly, N., Le, Q. V. et Vinyals, O. (2015). Listen, attend and spell. *arXiv preprint arXiv :1508.01211*.

[Che *et al.*, 2016] Che, T., Li, Y., Jacob, A. P., Bengio, Y. et Li, W. (2016). Mode regularized generative adversarial networks. *arXiv preprint arXiv :1612.02136*.

[Che *et al.*, 2017] Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y. et Bengio, Y. (2017). Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv :1702.07983*.

[Chelba *et al.*, 2013] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P. et Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv :1312.3005*.

[Cho *et al.*, 2014a] Cho, K., Van Merriënboer, B., Bahdanau, D. et Bengio, Y. (2014a). On the properties of neural machine translation : Encoder-decoder approaches. *arXiv preprint arXiv :1409.1259*.

[Cho *et al.*, 2014b] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. et Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv :1406.1078*.

[Cho *et al.*, 2014c] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H. et Bengio, Y. (2014c). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Chung *et al.*, 2016] Chung, J., Cho, K. et Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv :1603.06147*.

[Chung *et al.*, 2014] Chung, J., Gulcehre, C., Cho, K. et Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv :1412.3555*.

[Cooijmans *et al.*, 2016] Cooijmans, T., Ballas, N., Laurent, C. et Courville, A. C. (2016). Recurrent batch normalization. *CoRR*, abs/1603.09025.

[Dauphin *et al.*, 2016] Dauphin, Y. N., Fan, A., Auli, M. et Grangier, D. (2016). Language modeling with gated convolutional networks. *arXiv preprint arXiv :1612.08083*.

[Deng *et al.*, 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et Fei-Fei, L. (2009). ImageNet : A Large-Scale Hierarchical Image Database. *In CVPR09*.

[Dietterich, 2000] Dietterich, T. G. (2000). Hierarchical reinforcement learning.

[Dumoulin *et al.*, 2016] Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O. et Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv :1606.00704*.

[Earley, 1970] Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[Gauthier, 2014] Gauthier, J. (2014). Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N : Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014(5):2.

[Gehring *et al.*, 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D. et Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv :1705.03122*.

[Glorot *et al.*, 2011] Glorot, X., Bordes, A. et Bengio, Y. (2011). Deep sparse rectifier neural networks. *In Aistats*, volume 15, page 275.

[Goodfellow *et al.*, 2016] Goodfellow, I., Bengio, Y. et Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Goodfellow *et al.*, 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. et Bengio, Y. (2014). Generative adversarial nets. *In Advances in neural information processing systems*, pages 2672–2680.

[Graves et Schmidhuber, 2009] Graves, A. et Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. *In Advances in neural information processing systems*, pages 545–552.

[Gulcehre *et al.*, 2016] Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B. et Bengio, Y. (2016). Pointing the unknown words. *arXiv preprint arXiv :1603.08148*.

[Gulcehre *et al.*, 2017] Gulcehre, C., Chandar, S. et Bengio, Y. (2017). Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv :1701.08718*.

[Gulrajani *et al.*, 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. et Courville, A. (2017). Improved training of wasserstein gans. *arXiv preprint arXiv :1704.00028*.

[He *et al.*, 2016] He, K., Zhang, X., Ren, S. et Sun, J. (2016). Deep residual learning for image recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91:1.

[Hochreiter et Schmidhuber, 1997] Hochreiter, S. et Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hu *et al.*, 2017] Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R. et Xing, E. P. (2017). Controllable text generation. *arXiv preprint arXiv :1703.00955*.

[Hutchins, 2005] Hutchins, J. (2005). The history of machine translation in a nutshell. *Retrieved December*, 20:2009.

[Ioffe et Szegedy, 2015a] Ioffe, S. et Szegedy, C. (2015a). Batch normalization : Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Ioffe et Szegedy, 2015b] Ioffe, S. et Szegedy, C. (2015b). Batch normalization : Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv :1502.03167*.

[Jang *et al.*, 2016] Jang, E., Gu, S. et Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv :1611.01144*.

[Jordan, 1986] Jordan, M. (1986). Serial order : a parallel distributed processing approach. technical report, june 1985-march 1986. Rapport technique, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.

[Kalchbrenner *et al.*, 2016] Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A. et Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv :1610.10099*.

[Kim *et al.*, 2015] Kim, Y., Jernite, Y., Sontag, D. et Rush, A. M. (2015). Character-aware neural language models. *arXiv preprint arXiv :1508.06615*.

[Kingma et Ba, 2014] Kingma, D. P. et Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

[Kingma et Welling, 2013] Kingma, D. P. et Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv :1312.6114*.

[Klein et Manning, 2003] Klein, D. et Manning, C. D. (2003). A parsing : fast exact viterbi parse selection. *In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 40–47. Association for Computational Linguistics.

[Lamb *et al.*, 2016] Lamb, A. M., GOYAL, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C. et Bengio, Y. (2016). Professor forcing : A new algorithm for training recurrent networks. *In Advances In Neural Information Processing Systems*, pages 4601–4609.

[Larochelle, 2013] Larochelle (2013). Neural networks. University Lecture.

[Laurent *et al.*, 2016] Laurent, C., Pereyra, G., Brakel, P., Zhang, Y. et Bengio, Y. (2016). Batch normalized recurrent neural networks. *In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2657–2661. IEEE.

[Lee *et al.*, 2016] Lee, J., Cho, K. et Hofmann, T. (2016). Fully character-level neural machine translation without explicit segmentation. *arXiv preprint arXiv :1610.03017*.

[Li *et al.*, 2015] Li, Y., Tarlow, D., Brockschmidt, M. et Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv :1511.05493*.

[Lin, 2004] Lin, C.-Y. (2004). Rouge : A package for automatic evaluation of summaries. *In Text summarization branches out : Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

[Luo *et al.*, 2016] Luo, Y., Chiu, C.-C., Jaitly, N. et Sutskever, I. (2016). Learning online alignments with continuous rewards policy gradient. *arXiv preprint arXiv :1608.01281*.

[Luong et Manning, 2016] Luong, M.-T. et Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv :1604.00788*.

[Maddison *et al.*, 2016] Maddison, C. J., Mnih, A. et Teh, Y. W. (2016). The concrete distribution : A continuous relaxation of discrete random variables. *arXiv preprint arXiv :1611.00712*.

[Mao *et al.*, 2016] Mao, X., Li, Q., Xie, H., Lau, R. Y. et Wang, Z. (2016). Least squares generative adversarial networks. *arXiv preprint ArXiv :1611.04076*.

[Marcus *et al.*, 1993] Marcus, M. P., Marcinkiewicz, M. A. et Santorini, B. (1993). Building a large annotated corpus of english : The penn treebank. *Computational linguistics*, 19(2):313–330.

[Mcculloch et Pitts, 1943] Mcculloch, W. et Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147.

[Mikolov *et al.*, 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J. et Khudanpur, S. (2010). Recurrent neural network based language model. *In Interspeech*, volume 2, page 3.

[Minsky et Papert, 1969] Minsky, M. et Papert, S. (1969). *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

[Mirza et Osindero, 2014] Mirza, M. et Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv :1411.1784*.

[Mozer, 1992] Mozer, M. C. (1992). Induction of multiscale temporal structure. *In Advances in neural information processing systems*, pages 275–282.

[Nair et Hinton, 2010] Nair, V. et Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Nowozin *et al.*, 2016] Nowozin, S., Cseke, B. et Tomioka, R. (2016). f-gan : Training generative neural samplers using variational divergence minimization. *In Advances in Neural Information Processing Systems*, pages 271–279.

[Oord *et al.*, 2016] Oord, A. v. d., Kalchbrenner, N. et Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *arXiv preprint arXiv :1601.06759*.

[Papineni *et al.*, 2002] Papineni, K., Roukos, S., Ward, T. et Zhu, W.-J. (2002). Bleu : a method for automatic evaluation of machine translation. *In Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[Pascanu *et al.*, 2013a] Pascanu, R., Gulcehre, C., Cho, K. et Bengio, Y. (2013a). How to construct deep recurrent neural networks. *arXiv preprint arXiv :1312.6026*.

[Pascanu *et al.*, 2013b] Pascanu, R., Mikolov, T. et Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. *In International Conference on Machine Learning*, pages 1310–1318.

[Polyak, 1964] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[Radford *et al.*, 2015] Radford, A., Metz, L. et Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv :1511.06434*.

[Rajpurkar *et al.*, 2016a] Rajpurkar, P., Zhang, J., Lopyrev, K. et Liang, P. (2016a). Squad : 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv :1606.05250*.

[Rajpurkar *et al.*, 2016b] Rajpurkar, P., Zhang, J., Lopyrev, K. et Liang, P. (2016b). Squad : 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv :1606.05250*.

[Ranzato *et al.*, 2016] Ranzato, M., Chopra, S., Auli, M. et Zaremba, W. (2016). Sequence level training with recurrent neural networks. *Proc. of ICLR*.

[Rumelhart *et al.*, 1986] Rumelhart, D. E., Hinton, G. E. et Williams, R. J. (1986). Parallel distributed processing : Explorations in the microstructure of cognition, vol. 1. chapitre Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

[Rush *et al.*, 2015] Rush, A. M., Chopra, S. et Weston, J. (2015). A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv :1509.00685*.

[Salimans *et al.*, 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. et Chen, X. (2016). Improved techniques for training gans. *In Advances in Neural Information Processing Systems*, pages 2226–2234.

[Sennrich *et al.*, 2015] Sennrich, R., Haddow, B. et Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv :1508.07909*.

[Serdyuk *et al.*, 2017] Serdyuk, D., Ke, N. R., Sordoni, A., Pal, C. et Bengio, Y. (2017). Twin networks : Using the future as a regularizer. *CoRR*, abs/1708.06742.

[Silver *et al.*, 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Sutskever *et al.*, 2014a] Sutskever, I., Vinyals, O. et Le, Q. V. (2014a). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

[Sutskever *et al.*, 2014b] Sutskever, I., Vinyals, O. et Le, Q. V. (2014b). Sequence to sequence learning with neural networks. pages 3104–3112.

[Tolstikhin *et al.*, 2017] Tolstikhin, I., Bousquet, O., Gelly, S. et Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv :1711.01558*.

[University of Illinois at Urbana-Champaign *et al.*, 1988] University of Illinois at Urbana-Champaign, C. f. S. R., Development et Cybenko, G. (1988). *Continuous valued neural networks with two hidden layers are sufficient*.

[Van Den Oord *et al.*, 2016] Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. et Kavukcuoglu, K. (2016). Wavenet : A generative model for raw audio. *CoRR abs/1609.03499*.

[Vezhnevets *et al.*, 2016] Vezhnevets, A., Mnih, V., Agapiou, J., Osindero, S., Graves, A., Vinyals, O. et Kavukcuoglu, K. (2016). Strategic attentive writer for learning macro-actions. *In Advances in Neural Information Processing Systems*, pages 3486–3494.

[Waibel *et al.*, 1990] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. et Lang, K. J. (1990). Phoneme recognition using time-delay neural networks. *In Readings in speech recognition*, pages 393–404. Elsevier.

[Williams, 1992a] Williams, R. J. (1992a). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *In Reinforcement Learning*, pages 5–32. Springer.

[Williams, 1992b] Williams, R. J. (1992b). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[Williams et Zipser, 1989] Williams, R. J. et Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[Wiseman et Rush, 2016] Wiseman, S. et Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization. *CoRR*, abs/1606.02960.

[Wu *et al.*, 2016] Wu, Y., Burda, Y., Salakhutdinov, R. et Grosse, R. (2016). On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv :1611.04273*.

[Xu *et al.*, 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R. et Bengio, Y. (2015). Show, attend and tell : Neural image caption generation with visual attention. *In International Conference on Machine Learning*, pages 2048–2057.

[Yang *et al.*, 2016] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. et Hovy, E. (2016). Hierarchical attention networks for document classification. *In Proceedings of NAACL-HLT*, pages 1480–1489.

[Yu *et al.*, 2016] Yu, L., Zhang, W., Wang, J. et Yu, Y. (2016). Seqgan : sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv :1609.05473*.

[Yuan *et al.*, 2017] Yuan, X., Wang, T., Gulcehre, C., Sordoni, A., Bachman, P., Subramanian, S., Zhang, S. et Trischler, A. (2017). Machine comprehension by text-to-text neural question generation. *arXiv preprint arXiv :1705.02012*.

[Zaremba *et al.*, 2014] Zaremba, W., Sutskever, I. et Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv :1409.2329*.

[Zhang et Lapata, 2014] Zhang, X. et Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. *In EMNLP*, pages 670–680.

[Zhang *et al.*, 2015] Zhang, X., Zhao, J. et LeCun, Y. (2015). Character-level convolutional networks for text classification. *In Advances in Neural Information Processing Systems*, pages 649–657.

[Zhao *et al.*, 2018] Zhao, J., Kim, Y., Zhang, K., Rush, A. et LeCun, Y. (2018). Adversarially regularized autoencoders. *In* Dy, J. et Krause, A., éditeurs : *Proceedings of the 35th International Conference on Machine Learning*, volume 80 de *Proceedings of Machine Learning Research*, pages 5902–5911, Stockholmsmässan, Stockholm Sweden. PMLR.